通信技术与应用丛书

基于dsPIC的无线通信系统设计

Wireless Communication System Design Based on dsPIC

魏以民　姚铁　杨涛　张祯松　陈丽花　编著







- 理论完备：内容反映了无线通信原理的基础理论
- 案例经典：深入剖析经典案例展示无线通信中的典型应用
- 学习方便：学习门槛较低，受众广泛，人人可以动手一试




机械工业出版社
CHINA MACHINE PRESS

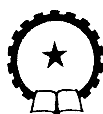
VISIT…

LANZAROTE Caliente.com

通信技术与应用丛书

基于 dsPIC 的无线通信系统设计

魏以民　姚　铁　杨　涛　张祯松　陈丽花　编著



机 械 工 业 出 版 社

本书采用美国微芯公司的 dsPIC 芯片来实现无线通信中的常见算法，并通过剖析一个典型案例来分析在业余无线电爱好者中广泛应用的 NUE-PSK 调制解调器，从而展示出 dsPIC 在无线通信中的典型应用。本书具有学习门槛低、受众广泛明确等特点。

本书分 7 章，内容包括数字信号控制器及其在无线通信中的应用、MPLAB C30 编译器、数字滤波器的设计与实现、数字调制解调器的设计与实现、同步功能的设计与实现、信道编/译码器的设计与实现、基于 dsPIC 的无线通信设备 NUE-PSK 实例剖析。

本书可以作为通信与信息系统专业研究生、本科生的参考书，也可供通信工程技术人员参考。

图书在版编目（CIP）数据

基于 dsPIC 的无线通信系统设计 / 魏以民等编著．—北京：机械工业出版社，2011.12

（通信技术与应用丛书）

ISBN 978-7-111-36505-1

Ⅰ．①基… Ⅱ．①魏… Ⅲ．①无线电通信－通信系统－系统设计

Ⅳ．①TN92-39

中国版本图书馆 CIP 数据核字（2011）第 238533 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

责任编辑：李馨馨

责任印制：杨 曦

北京圣夫亚美印刷有限公司印刷

2012 年 1 月第 1 版 • 第 1 次印刷

184mm×260mm • 12.75 印张 • 312 千字

0001—3000 册

标准书号：ISBN 978-7-111-36505-1

定价：32.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务

社服务中心：（010）88361066

销 售 一 部：（010）68326294

销 售 二 部：（010）88379649

读者购书热线：（010）88379203

网络服务

门户网：http://www.cmpbook.com

教材网：http://www.cmpedu.com

封面无防伪标均为盗版

前　言

近年来，无线通信以其方便快捷的特点受到重视，得到了飞速发展，学习和研究无线通信的学生和科研人员也越来越多。与此同时，无线通信的实现方法也发生了翻天覆地的变化，从最开始的模拟通信系统到数字通信系统，以至于到最新的基于软件无线电的实现方法。目前大部分的无线通信中的算法都是基于 DSP 芯片，用软件的方法来实现。

目前市场上 DSP 主要有 TI、AD 和 MOTOROLA 公司的产品，但是随着 Microchip 公司在单片机市场上的异军突起，Microchip 公司也把 PIC 单片机和 DSP 的特点结合起来，推出了 dsPIC 系列芯片。该系列芯片继承了 PIC 单片机稳定、高效的特点，并将常用的 DSP 指令集成进来，如乘累加（MAC）、位反转寻址、单周期最多 16 位左移或右移操作、单指令循环等。这使得 dsPIC 芯片不仅在控制领域有着广阔的应用，而且可以在计算密集型的领域，如通信、信号处理领域大有可为。

作为 PIC 系列单片机的扩展，dsPIC 的用户群具有天然的优势和极强的渗透力，可以预见 dsPIC 必将在通信和信号处理领域得到广泛的应用。本书就是基于这个预期，利用 dsPIC 来实现通信中的常见算法，并通过剖析一个典型案例的方法来分析一个在业余无线电爱好者中广泛应用的 NUE-PSK 型调制解调器，从而展示出 dsPIC 在无线通信中的典型应用。

本书第 1 章介绍目前常用的 dsPIC33F 系列的数字信号控制器，包括系统结构、最小系统和利用 dsPIC33F 处理器实现的一个无线通信设备。第 2 章介绍了 dsPIC33F 处理器采用的 MPLAB C30 标准的特点，以及其和 ANSI C 的区别。第 3 章介绍了数字滤波器的设计与实现，包括 IIR 滤波器的设计与实现，FIR 滤波器的设计与实现，以及用 dsPIC33F 来实现 IIR 滤波器和 FIR 滤波器的实例。第 4 章介绍了数字调制解调功能的设计与实现，包括数字调制的设计与实现，数字解调的设计与实现，以及数字调制解调在 dsPIC33F 上实现的实例。第 5 章介绍了同步功能的设计与实现，包括载波同步、位同步和帧同步的设计与实现，以及在 dsPIC33F 实现的实例。第 6 章介绍了信道编译码的设计与实现，包括线性分组码的原理与实现，卷积码的设计与实现，以及在 dsPIC33F 上实现卷积码的实例。第 7 章详细介绍了一个采用 dsPIC 来实现的无线通信设备 NUE-PSK，分别介绍了其软件和硬件的详细实现细节。附录介绍了业余无线电的简单情况。

本书具有如下特点：（1）门槛低。与 TI 等 DSP 开发系统动辄几千上万的开发系统相比，dsPIC 的开发系统的价格非常低廉，一般只有一两百元，并且和 PIC 单片机的开发系统通用。这样就大大方便了读者来动手尝试一下书中的算法和程序。（2）受众明确。PIC 单片机培养了大批 PIC 单片机的开发者。他们有进一步利用数字信号处理技术来提高产品档次和性能的需求，dsPIC 提供了这样的一个机会。本书就是为 PIC 单片机的用户量身定做的。（3）典型案例剖析。本书通过剖析一个典型案例来分析一个在业余无线电中广泛应用的 NUE-PSK 型调制解调器，从而展示了 dsPIC 在无线通信和信号处理中的典型应用。读者可以通过对例子中算法的分析，进一步掌握 dsPIC 芯片在常用的数字信号处理算法中的应用方法。

本书由姚轶编写第 1 章和第 7 章的 7.1、7.2 节，张祯松编写第 2 章，陈丽花编写第 3

章，杨涛编写第 4 章和第 5 章的 5.1、5.2 节，魏以民编写第 5 章的 5.3 节、第 6 章、第 7 章的 7.3～7.5 节和附录。

本书得到自然科学基金项目“基于无线自组网的应急通信关键技术问题研究（编号：61072043）”支持。

目　录

前言
第 1 章　数字信号控制器及其在无线通信中的应用……1
1.1　数字信号控制器……1
1.1.1　dsPIC33F 系列数字信号控制器简介……1
1.1.2　dsPIC33F 系列数字信号控制器的系统结构……3
1.2　CPU 模块……4
1.2.1　内部寄存器……4
1.2.2　DSP 引擎……5
1.2.3　数据存储器的控制……7
1.2.4　程序存储器的控制……8
1.2.5　中断机制……8
1.3　外设模块……9
1.3.1　A/D 转换器……9
1.3.2　通用定时模块……10
1.3.3　输入捕捉模块……12
1.3.4　输出比较模块……13
1.3.5　SPI 模块……14
1.3.6　UART 接口模块……15
1.3.7　I2C 模块……18
1.3.8　I/O 引脚……19
1.4　dsPIC33F 系列数字信号控制器构成的最小系统……19
1.4.1　时钟振荡器控制电路……19
1.4.2　复位电路……20
1.4.3　看门狗定时器电路……21
1.4.4　低功耗电源管理电路……22
1.5　无线通信中的数字信号处理技术……23
1.5.1　数字滤波器……24
1.5.2　数字调制技术……25
1.5.3　同步控制技术……29
1.5.4　差错控制技术……32
1.6　dsPIC33F 系列数字信号控制器在无线通信中的实例……35
第 2 章　MPLAB C30 编译器……37
2.1　MPLAB C30 与 ANSI C 的差别……37
2.1.1　关键字差别……37
2.1.2　语句差别……48
2.1.3　表达式差别……49
2.2　MPLAB C30 C 编译器运行时环境……50
2.2.1　地址空间……50
2.2.2　代码段和数据段……50
2.2.3　启动和初始化……51
2.2.4　存储空间……52
2.2.5　存储模型……53
2.2.6　定位代码和数据……54
2.2.7　软件堆栈……55
2.2.8　C 编译器中堆栈的使用……56
2.2.9　C 编译器中堆的使用……57
2.2.10　函数调用约定……58
2.2.11　寄存器约定……59
2.2.12　位反转寻址和模寻址……60
2.2.13　程序空间可视性的使用……60
第 3 章　数字滤波器的设计与实现……61
3.1　数字滤波的基本概念……61
3.1.1　时域离散信号……61
3.1.2　线性时不变系统……61
3.1.3　卷积……63
3.1.4　数字滤波器的基本概念……64
3.2　IIR 滤波器……65
3.2.1　IIR 滤波器的基本原理和设计方法……65
3.2.2　IIR 滤波器的 MATLAB

设计 ……………………………………… 65
3.2.3 IIR 滤波器的实例 ………………… 72
3.3 FIR 滤波器……………………………73
3.3.1 FIR 滤波器的基本原理和设计方法 …………………… 73
3.3.2 FIR 滤波器的 MATLAB 实现 ……………………………………… 73
3.3.3 几种重要的 MATLAB 滤波器的设计参数……………… 76
3.3.4 FIR 滤波器的 DSP 实现……… 78
第 4 章 数字调制解调器的设计与实现 ………………………… 83
4.1 无线通信中的数字调制 …………83
4.1.1 无线通信系统对数字调制的要求……………………… 83
4.1.2 数字信号的带宽和功率谱密度……………………… 84
4.2 调制……………………………………85
4.2.1 调制器的基本原理 ……………… 85
4.2.2 脉冲成型的设计………………… 87
4.2.3 调制器的 MATLAB 仿真…………………………………… 90
4.2.4 调制器的 dsPIC 实现…………… 94
4.3 解调…………………………………101
4.3.1 解调器的基本原理 ……………101
4.3.2 解调器的 MATLAB 仿真 ……………………………………103
4.3.3 解调器的 dsPIC 实现…………105
第 5 章 同步功能的设计与实现……… 109
5.1 载波同步…………………………109
5.1.1 载波同步的基本原理…………110
5.1.2 载波同步的 MATLAB 实现…………………………………111
5.1.3 载波同步的 dsPIC 实现 ………112
5.2 位同步……………………………114
5.2.1 位同步的方法…………………115
5.2.2 位同步的 MATLAB 实现 ……………………………………120
5.2.3 位同步的 dsPIC 实现…………121
5.3 帧同步……………………………122
5.3.1 群同步的方法 …………………… 122
5.3.2 帧同步编码的 MATLAB 实现 ……………………………… 124
5.3.3 帧同步编码的 dsPIC 实现 ………………………………… 125
5.3.4 帧同步解码的 dsPIC 实现 ………………………………… 128
第 6 章 信道编译码器的设计与实现……………………………… 130
6.1 概述………………………………………130
6.2 线性分组码原理及实现 ………132
6.2.1 线性分组码的基本原理……… 132
6.2.2 （7，4）汉明码的 MATLAB 实现 …………………… 136
6.3 卷积码原理及其实现 ……………137
6.3.1 卷积码的基本原理……………… 137
6.3.2 卷积码编译码的 MATLAB 实现 …………………… 140
6.3.3 卷积码的 dsPIC 实现 ………… 140
第 7 章 基于 dsPIC 无线通信设备 NUE-PSK31 型数字调制解调器实例剖析 ………………………………… 143
7.1 PSK31 型数字调制解调器简介 …………………………143
7.2 PSK31 型数字调制解调器人机交互接口 ……………………………146
7.2.1 PS/2 键盘输入接口 …………… 146
7.2.2 旋转编码器输入接口………… 148
7.2.3 LCD 显示接口 ………………… 150
7.2.4 数/模与模/数接口……………… 153
7.2.5 模/数接口 ………………………… 154
7.2.6 数/模接口 ………………………… 161
7.2.7 I2C 外部存储接口……………… 163
7.3 软件程序概况……………………167
7.4 发送端软件 ………………………167
7.4.1 可变长编码 ……………………… 168
7.4.2 BPSK/QPSK 串/并转换和差分编码 ……………………………… 168

7.4.3 成型滤波和调制……170
7.4.4 数/模转换……171
7.5 接收端软件……171
7.5.1 模/数转换……173
7.5.2 512 点 FFT……173
7.5.3 解调……174
7.5.4 抽取滤波器……175
7.5.5 比特匹配滤波器……176
7.5.6 频率滤波器……176
7.5.7 AGC……177
7.5.8 载波同步……178
7.5.9 位同步……180
7.5.10 差分译码……182
7.5.11 软判决维特比译码……182
7.5.12 可变长信源译码……184
7.5.13 静噪控制和信号质量计算……184
7.6 DSP 库简介……187
附录 业余无线电简介……189
参考文献……195

第 1 章　数字信号控制器及其在无线通信中的应用

1.1　数字信号控制器

1.1.1　dsPIC33F 系列数字信号控制器简介

随着微电子技术的迅猛发展，单片机不断向高速化、多样化方向发展，其功能越来越强、成本越来越低。近年来，美国微芯（Microchip）公司推出了 dsPIC33F 系列高性能 16 位数字信号控制器（DCS），其性价比介于 16 位、32 位单片机及 DSP 中、低档机之间。目前在智能控制、自动检测、无线通信等领域正掀起了广泛使用 DSP 的热潮。DSP 即数字信号处理器，是在数字信号处理的各种理论和算法的基础上发展起来的，用于完成各种实时数字信息处理的 CPU。20 世纪 80 年代初，随着大规模集成电路技术的发展，DSP 也得到了广泛的应用。DSP 器件的出现使得各种数字信号处理的算法得到了实现。DSP 器件不但使数字信号处理得到了实际应用，而且还拓宽到了系统控制领域，从而诞生了一大批新型的电子产品。DSP 技术的迅速普及，也为今天的信息高速公路建设奠定了基础。

但是 DSP 器件也有自己的缺点，就是控制能力相对较弱，芯片所提供的接口种类和数量较少，所以在实际系统中，人们通常把控制功能交给单片机（MCU）处理，而把复杂的算法或运算量大的部分交给数字信号处理器（DSP），这样一来一个系统就需要两个微处理器，系统的复杂性和成本提高了，而系统的可靠性和效率却下降了。为解决这一矛盾，微芯公司的 dsPIC33F 系列数字信号控制器将高性能 16 位单片机的控制特点和 DSP 高速运算的优点结合起来，为嵌入式系统设计提供了适合单芯片、单指令流的解决方案。它消除了目前类似设计中所需求的额外组成部分，既减小了印制板空间，也降低了系统成本，同时 dsPIC33F 系列数字信号控制器的成本也较为低廉，由其构成的智能设备和系统具有较高的性价比。

（1）数字信号控制器的特点

dsPIC33F 系列数字信号控制器兼容了 PIC 单片机和 DSP 器件两类产品的优点，它具有以下特点：

- 具有丰富的外围接口部件。
- 具有丰富的片内资源。
- 具有完整的 DSP 引擎。
- 可通过内部的 Flash 存储器灵活地保存程序。

● 中断控制器可确保外部事件及时响应。

● 具有以优化的高级语言为基础的强大的开发环境。

● 系列芯片种类多，可选范围大。

（2）数字信号控制器的分类

dsPIC33F 系列数字信号控制器有多种分类方法，从功能上分可分为通用系列和控制系列，两者的区别是电动机控制系列的 DCS 拥有电动机控制；根据芯片引脚数的不同可分为 64 引脚、80 引脚和 100 引脚三种；根据片内 Flash 存储器容量的不同分可分为 64KB、128KB 和 256KB 三种；根据片内 RAM 存储器容量的不同可分为 8KB、16KB 和 30KB 三种。

（3）数字信号控制器的命名规则

dsPIC33F 系列数字信号控制器的命名规则如图 1−1 所示。

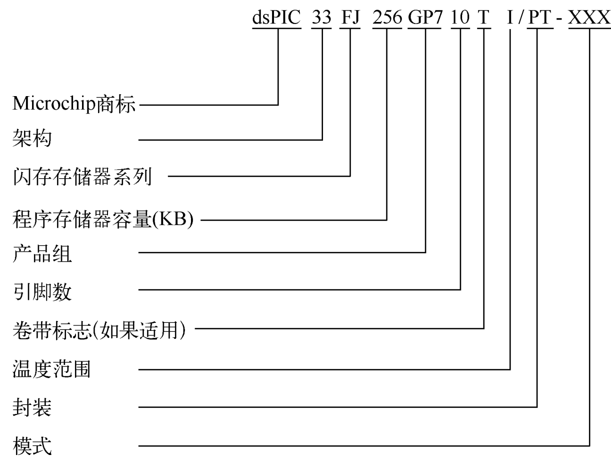


图 1−1　dsPIC33F 系列数字信号控制器的命名规则

图中，DCS架构“33”代表是 33 系列数字信号控制器；闪存存储器系列“FJ”代表是 3.3V的闪存程序存储器；产品组“GP7”代表是通用系列数字信号控制器（此外还包括通用系列GP2、GP3、GP5 和电动机控制系列MC5、MC7）；引脚数“10”代表芯片的引脚数是 100（此外“06”代表 64 脚，“08”代表 80 脚）；温度范围“I”表示工业级（−40～85℃）；封装“PT”代表是 10×10mm^{2}或 12×12mm^{2}的 TQFP（Thin Quad Flat Pack，薄型四方扁平）封装；模式“XXX”是三位数字，表示QTP、SQTP（Microchip公司在美国的服务标记）、编码或特殊要求。

（4）dsPIC33F 系列数字信号控制器的引脚功能

dsPIC33F 系列数字信号控制器的引脚功能如图 1−2 所示。

dsPIC33F 系列数字信号控制器的用途非常广泛，例如电动机控制、互联网接入、汽车控制产品、多功能电话、数字应答机、低速软件调制解调器、线卡、POS 终端、自动售货机、生物测定安全装置、不间断电源、电源管理和自然语音输入/输出等。

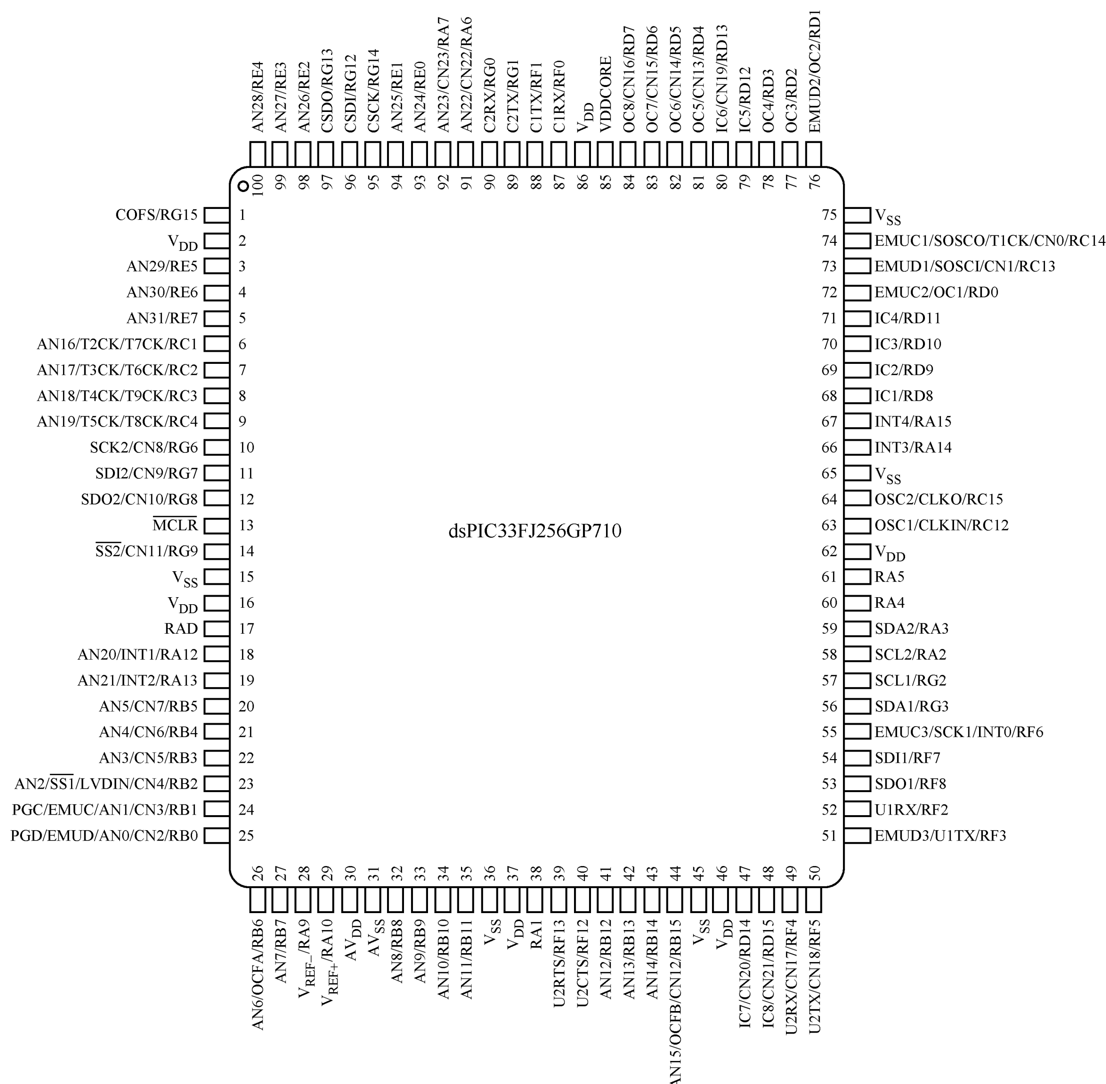


图 1-2　dsPIC33F 系列数字信号控制器的引脚功能图

1.1.2　dsPIC33F 系列数字信号控制器的系统结构

dsPIC33F 系列数字信号控制器内部由 CPU 模块和外设模块两部分组成。CPU 模块包括一个 16 位的单片机、一个高速 DSP 引擎、一组支撑单片机和 DSP 运转的若干内部寄存器、一定容量的数据存储区和程序存储区以及灵活可靠的中断机制；外设模块包括 A/D 转换器、通用定时器模块、电机控制 PWM 模块、正交编码器接口（QEI）模块、UART 模块、SPI 模块、CAN 总线模块等。dsPIC33F 系列数字信号控制器的内部结构如图 1-3 所示。

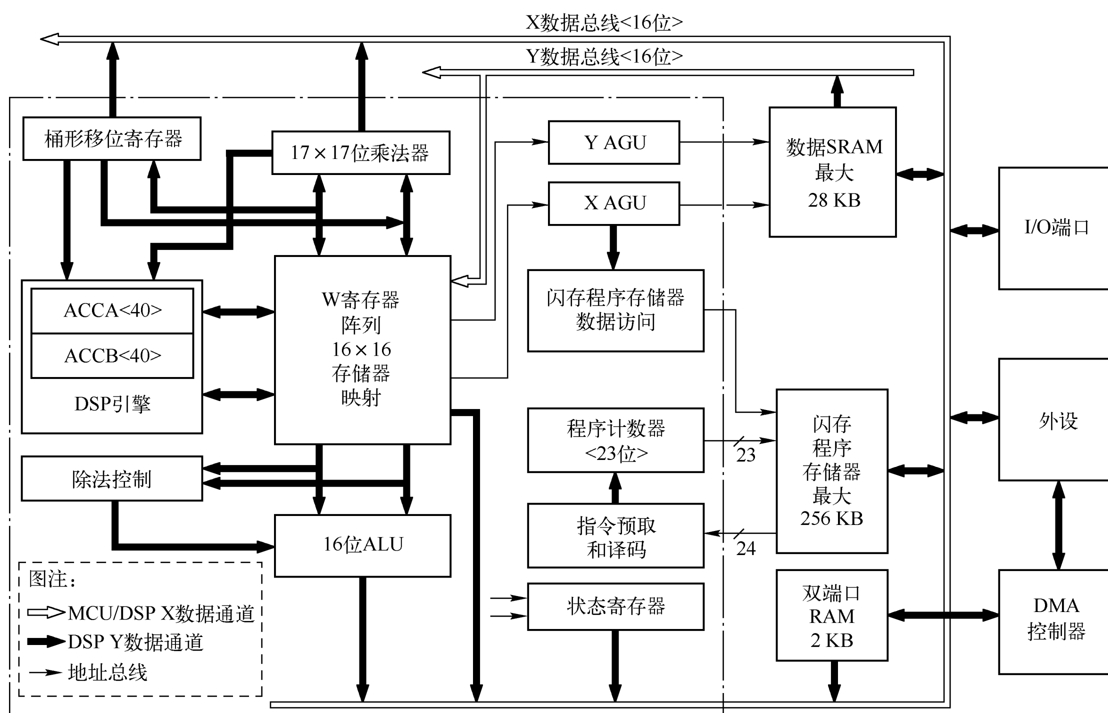


图 1-3　dsPIC33F 系列数字信号控制器的内部结构

1.2　CPU 模块

dsPIC33F 的 CPU 模块采用 16 位数据总线的改进型哈佛结构，具有增强指令集，其中包含了对 DSP 指令的支持。大部分指令都是单周期指令，并通过单周期指令预取机制提高程序运行时的吞吐量，并可提供可预测执行能力。CPU 拥有 24 位指令字，指令字带有长度可变的操作码字段。程序计数器（Program Counter，PC）为 23 位宽，可以寻址最大 4M×24 位的用户程序存储空间。

1.2.1　内部寄存器

dsPIC33F 的内部寄存器由 16 个 16 位工作寄存器（W0 ～ W15）、两个 40 位累加器（ACCA 和 ACCB）、状态寄存器（SR）、内核控制寄存器（CORCON）、数据表页寄存器（TBLPAG）、程序空间可视性页寄存器（PSVPAG）、DO 和 REPEAT 寄存器（DOSTART、DOEND、DCOUNT 和 RCOUNT）以及程序计数器（PC）组成。

（1）工作寄存器

工作寄存器可作为数据、地址或偏移量寄存器，其中 W0 是唯一能够在执行文件寄存器寻址的工作寄存器。W15 是专用的软件堆栈指针（Stack Pointer，SP），在程序异常处理、子程序调用和返回时 W15 会被自动修改。W14 是专用的堆栈帧指针，由 LNK 和 ULNK 指令定义，它也能够被任何指令引用（方式与引用所有其他工作寄存器相同）。堆栈指针总是指向第一个可用的字并从低地址到高地址生长。出栈（读操作）时，堆栈指针预递减，压栈

（写操作）时，堆栈指针后递增。工作寄存器的内部结构如图 1-4 所示。

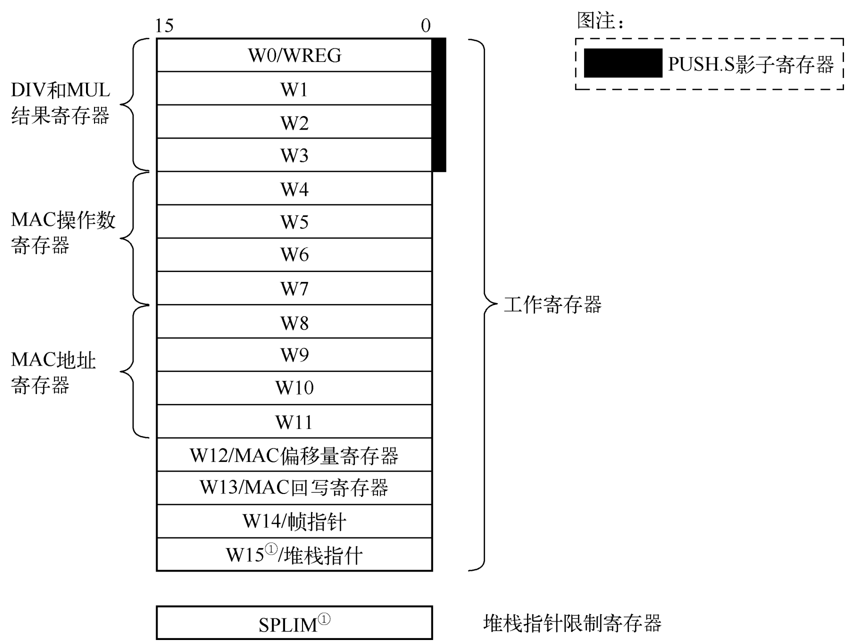


图 1-4　工作寄存器的内部结构

① W15 和 SPLIM 没有影子寄存器

一些寄存器有与之相关联的影子寄存器。影子寄存器是用来临时保存数据的寄存器，当一个周期内发生某些事件时它会将主寄存器的内容备份在影子寄存器中，也可以将影子寄存器的内容送回主寄存器。影子寄存器都是不可直接访问的。当某个工作寄存器执行字节操作时，只有目标寄存器的低字节会受到影响。所有寄存器都是存储器映射的，对于存储器映射的工作寄存器来说，可以通过对数据存储空间进行字节宽度的访问来对工作寄存器的低字节和高字节进行操作。

（2）状态寄存器

状态寄存器（SR）是一个 16 位的寄存器，它分成高、低字节的寄存器，低字节寄存器称为低状态寄存器（Lower Status Register，SRL），高字节寄存器称为高状态寄存器（Higher Status Register，SRH）。SRL 包含了所有的微控制单元（MCU）、算术逻辑单元（ALU）的操作状态标志，以及 CPU 中断优先级状态位 IPL<2：0>和循环有效状态位 RA；SRH 包含了 DSP 加法器/ 减法器状态位、DO 循环有效位 DA 和辅助进位标志位 DC 等。

（3）内核控制寄存器

内核控制寄存器（CORCON）包含 DSP 乘法器和 DO 循环硬件操作的控制位，还包含 CPU 中断优先级状态位 IPL3。

1.2.2　DSP 引擎

DSP 引擎由一个高速的单周期 17×17 位乘法器、一个桶形移位寄存器、一个带有两个目标累加器的 40 位加法器/ 减法器以及舍入与溢出逻辑组成，该模块的输入数据来自于工作寄存器 W4、W5、W6、W7 或者 X 存储器区，输出数据送到 DSP 指令定义的目标累加器

中或者 X 存储器区中。DSP 指令可以无缝地与所有其他指令一起操作，其设计可实现最佳的实时性能。MAC 指令和其他相关指令可在同一个周期内，同时完成从存储器中取出两个数据操作数、将两个工作寄存器相乘并进行累加的操作。DSP 引擎能够执行固有的“累加器—累加器”的操作而无需额外数据，这些指令包括 ADD、SUB 和 NEG 等。该功能大大简化了 32 位或 40 位数据的算术运算。DSP 引擎的内部结构如图 1−5 所示。

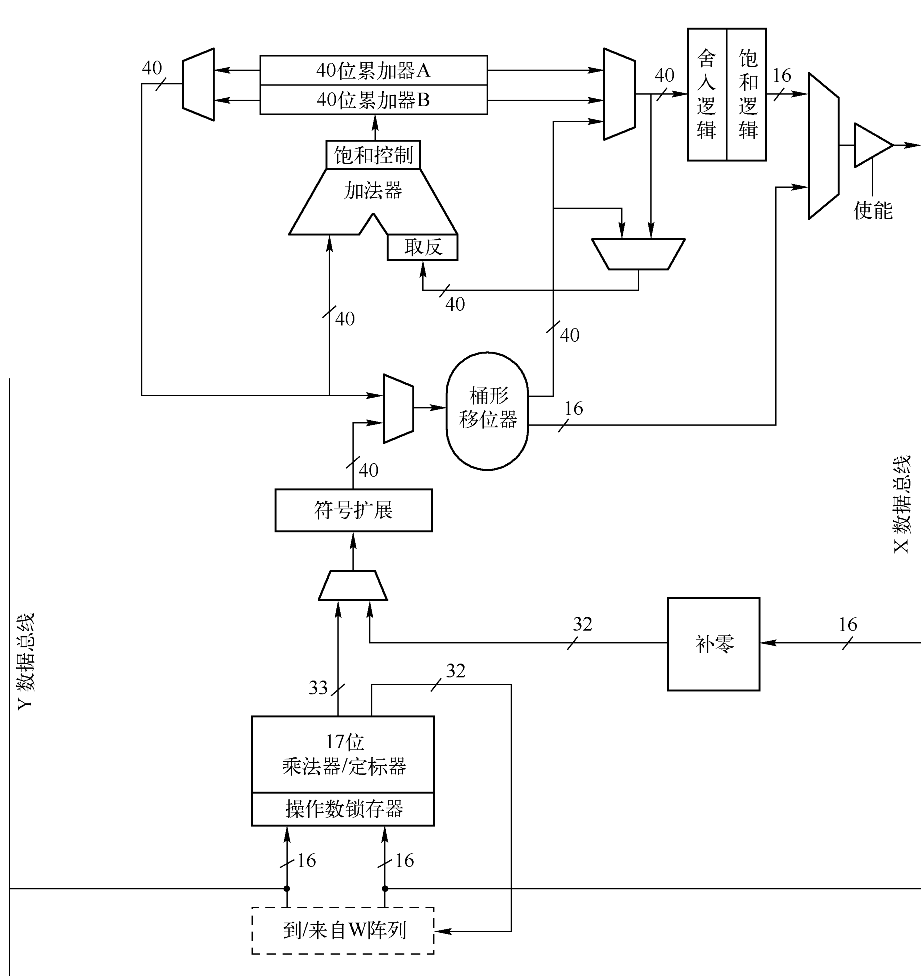


图 1−5 DSP 引擎的内部结构图

17×17 位乘法器可进行有符号或无符号运算。对输出进行适当调整可以支持 1.31 小数（Q31）或 32 位整数结果，无需在运算后手工处理小数乘法运算的结果。

40 位加法器/减法器可以选择两个累加器（A 或 B）之一作为保持运算后的结果，对于 ADD 指令和 LAC 指令，可选择通过桶形移位器在累加之前对将被累加或装入的数据进行调整。加法器/减法器产生溢出状态位 SA/SB 和 OA/OB，这些位在状态寄存器中，它也可以选择性地产生算术错误陷阱。

桶形移位寄存器在单个周期内可完成 16 位的算术右移或左移。 DSP 指令或 MCU 指令都可实现桶形移位器的多位移位。移位器需要一个带符号的二进制值确定移位操作的幅度（位数）和方向，具体规则如下：

- 若为正值则将操作数右移。
- 若为负值则将操作数左移。
- 若值为“0”则不改变操作数。

桶形移位器的宽度与累加器相同，都是 40 位，为 DSP 移位操作提供了 40 位输出结果，为 MCU 移位操作提供 16 位结果。

DSP 引擎还有一个舍入与溢出逻辑控制模块，可以控制累加器的数据饱和。该模块通过加法器的运算结果、状态位和用户设置的控制位来决定运算是否溢出，以及什么值会溢出，另外该模块还可以判断对数据空间进行写操作的溢出。

1.2.3　数据存储器的控制

dsPIC33F 的数据存储空间由两个 16 位的数据区组成，分别是 X、Y 数据空间。数据存储器 0x0000～0x07FF 之间的地址空间为保留空间，用于器件的特殊功能寄存器（SFR），包含了 CPU 和器件上所有外设的控制和状态位。X、Y 数据空间从地址 0x0800 开始，其空间划分视不同器件而定。对于数据写操作，总是将 X、Y 数据空间作为一个线性数据空间访问。对于数据读操作，可以分别单独访问 X、Y 存储器空间或将它们作为一个线性空间访问。用 MCU 类指令进行的数据读操作总是将 X、Y 数据空间作为一个组合的数据空间访问。而 DSP 指令通常具有两个源操作数（如 MAC 指令），可分别单独访问 X、Y 数据空间以支持同时对这两个源操作数进行读操作。这样 MCU 指令可以使用任何工作寄存器作为地址指针进行读/写操作，而 DSP 指令可以同时预取指两个数据操作数，将数据存储器分割为两个空间，必须使用指定的 W 寄存器作为读操作的地址指针。数据存储空间的分布如图 1−6 所示。

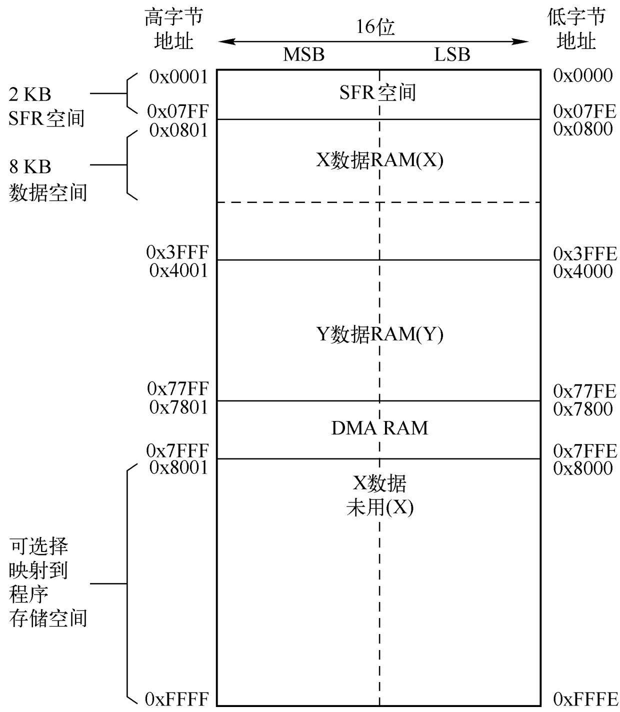


图 1−6　数据存储空间的分布

在数据存储空间的 0x0000～0x1FFF 之间保留了一个 8KB 的地址空间（称为 Near 数据存储器）。可通过所有文件寄存器指令中的 13 位绝对地址字段直接对 Near 数据存储器寻址。Near 数据区域中所包含的存储器区域由各个不同器件所实现的数据存储器的数量决定。Near 数据存储区域至少将包含所有 SFR 和某些 X 数据存储器。对于具备较大数据存储空间的器件，Near 数据区域可能包括所有 X 存储空间和部分（或全部）Y 存储空间。

为提高 DSP 算法中反复循环代码的效率，dsPIC33F 提供了循环寻址功能，该功能使用硬件自动支持循环数据缓冲区的方法，不需要用软件执行数据地址边界检查。除 W15 以外的任何工作寄存器都可以作为指向模缓冲区的指针，模寻址的硬件在所选工作寄存器保存的地址上执行边界检查，并在需要时自动调整指向缓冲区边界的指针值。

1.2.4　程序存储器的控制

dsPIC33F 有 4M×24 位程序存储空间。对程序存储空间的访问有以下三种途径：

- 通过 23 位 PC 程序计数器访问。
- 通过读表（TBLRD）和写表（TBLWT）指令访问。
- 通过把程序存储器的 32KB 段映射到数据存储器地址空间访问。

程序存储器映射空间被划分为用户程序空间和用户配置空间。用户程序空间包含复位向量、中断向量表、程序存储器和数据 EEPROM 存储器。用户配置空间包含用于设置器件选项的非易失性配置位和器件 ID 单元。

1.2.5　中断机制

dsPIC33F 具有 5 个非屏蔽陷阱源和 67 个中断源，为每个中断源分配 7 个等级的优先级。中断向量表（IVT）位于程序存储器中，起始单元地址是 0x000004。每个中断向量都包含 24 位地址，保存中断服务程序（ISR）的起始地址。CPU 负责读取中断向量表并将中断向量中的地址传递给程序计数器。

中断处理需要 4 个指令周期。第 1 个指令周期为外设中断发生后的指令周期，在该指令周期中完成当前指令操作并将中断标志状态位置 1；第 2 个指令周期中 PC 和 SRL 寄存器的内容被存入临时缓冲寄存器，并执行一个 NOP 操作，以保持与双周期指令中所进行的序列的一致性；在第 3 个周期中，PC 被装入中断源的向量表地址并将程序跳转到 ISR 的起始地址；在第 4 个周期中，PC 被装入 ISR 地址，并执行 NOP 操作。然后程序运行 ISR 地址中的内容。

与中断处理有关的寄存器包括：

1）状态寄存器（SR）和内核控制寄存器（CORCON）：其中的 IPL<2：0>和 IPL3 一起表示中断优先级状态位。

2）中断控制寄存器 1（INTCON1）：定义了中断嵌套禁止（NSTDIS）位，以及用于处理陷阱源的控制和状态标志。

3）中断控制寄存器 2（INTCON2）：控制外部中断请求信号的行为和备用向量表的使用。

4）中断标志状态寄存器（IFSx）：对应每个中断源的中断请求标志，其中“x”表示寄存器编号。每个中断源都有一个状态位，它们由各自的外设和外部信号置 1，并通过软

件清零。

5）中断允许控制寄存器（IECx）：对应每个中断源的中断允许控制位，其中“x”表示寄存器编号。这些控制位用于分别允许来自外设或外部信号的中断。

6）中断优先级控制寄存器（IPCx）：为每个中断源设置中断优先级。

在默认情况下中断是可嵌套的，任何正在处理的 ISR 都可以被另一个具有更高用户分配优先级的中断源中断。通过将 NSTDIS 位（INTCON1<15>）置 1 禁止中断嵌套。当 NSTDIS 位被置 1 时，所有处理中的中断将通过设置 IPL<2：0>=111 强制 CPU 的优先级为 7，此时将有效屏蔽其他所有中断源，直到执行 RETFIE 指令。当中断嵌套被禁止时，用户分配的中断优先级无效，除非是为了解决同时等待处理的中断之间的冲突。当禁止中断嵌套时，IPL<2：0> 位变成只读，这将防止用户软件将 IPL<2：0> 设置为一个较低的值，从而防止其有效地重新使能中断嵌套。

对于中断的配置通常采用如下步骤：

1）根据是否需要中断嵌套来设置 NSTDIS 控制位（INTCON1<15>）。

2）通过写相应的 IPCx 控制寄存器中的控制位选择中断源的用户分配优先级，优先级取决于特定的应用和中断源类型。

3）在相关的 IFSx 状态寄存器中清零与外设相关的中断标志状态位。

4）通过在相应的 IECx 控制寄存器中置 1 打开与中断源相关的中断允许控制位，使能中断源。

1.3 外设模块

dsPIC33F 的强大功能主要来自于其内置的功能众多的外设模块。

1.3.1 A/D 转换器

dsPIC33F 提供带有单端和差分输入的最多 32 路模拟输入通道，这些模块提供片内多路开关和采样保持电路，用于实现高精度和高速的 A/D 转换。dsPIC33F 提供两种 A/D 转换器，一种是 10 位的高速 A/D 转换器，其采样速率可达 1.1Msps；另一种是 12 位高精度 A/D 转换器，其采样速率为 500ksps。每个 A/D 转换器内部最多有 4 个片内采样保持放大器，可同时对 2 路、4 路或 8 路模拟输入进行采样。采用的参考电压可以使用芯片内部电压，也可以使用外部的 V_{REF+}和 V_{REF-}引脚电压。A/D 转换器的内部结构如图 1−7 所示。

dsPIC33F 对 A/D 转换器的控制用到了 6 个控制状态寄存器，分别是

- ADCON1：A/D 控制寄存器 1。
- ADCON2：A/D 控制寄存器 2。
- ADCON3：A/D 控制寄存器 3。
- ADCHS：A/D 输入通道选择寄存器。
- ADPCFG：A/D 端口配置寄存器。
- ADCSSL：A/D 输入扫描选择寄存器。

其中，ADCON1、ADCON2 和 ADCON3 寄存器用来控制 A/D 转换器的工作；ADCHS 寄存器用于选择连接到采样保持放大器的输入引脚；ADPCFG 寄存器用于将模拟输入引脚配

置为模拟输入或数字 I/O；ADCSSL 寄存器用于选择要被扫描的输入顺序。

A/D转换器的采样时钟来自于器件的指令时钟或内部RC时钟源，采样最高速率由模拟模块时钟T_{AD}控制。一次A/D 转换需要 12 个时钟周期（12T_{AD}）。A/D转换器的输出保存在结果缓冲器中，这是一个 16 字的双口RAM，称为ADCBUF，16 个ADCBUF分别为ADCBUF0、ADCBUF1、ADCBUF2、…、ADCBUFE和ADCBUFF。该数据接口可以使用DMA方式进行高速数据传输。当电源电压V_{DD}和参考电压V_{REF}满足V_{DD}=5×(1±10%)V，且V_{REF}=V_{DD}时，A/D转换的全范围误差小于±1LSB（Least Significant Bit，最低有效位）。如果V_{DD}小于 5V或者V_{REF}小于V_{DD}，则精度会降低。A/D转换器可以在程序中手动启动，也可以通过触发方式启动。触发方式有 4 种：自动方式、Timer3、外部中断和PWM 周期匹配。

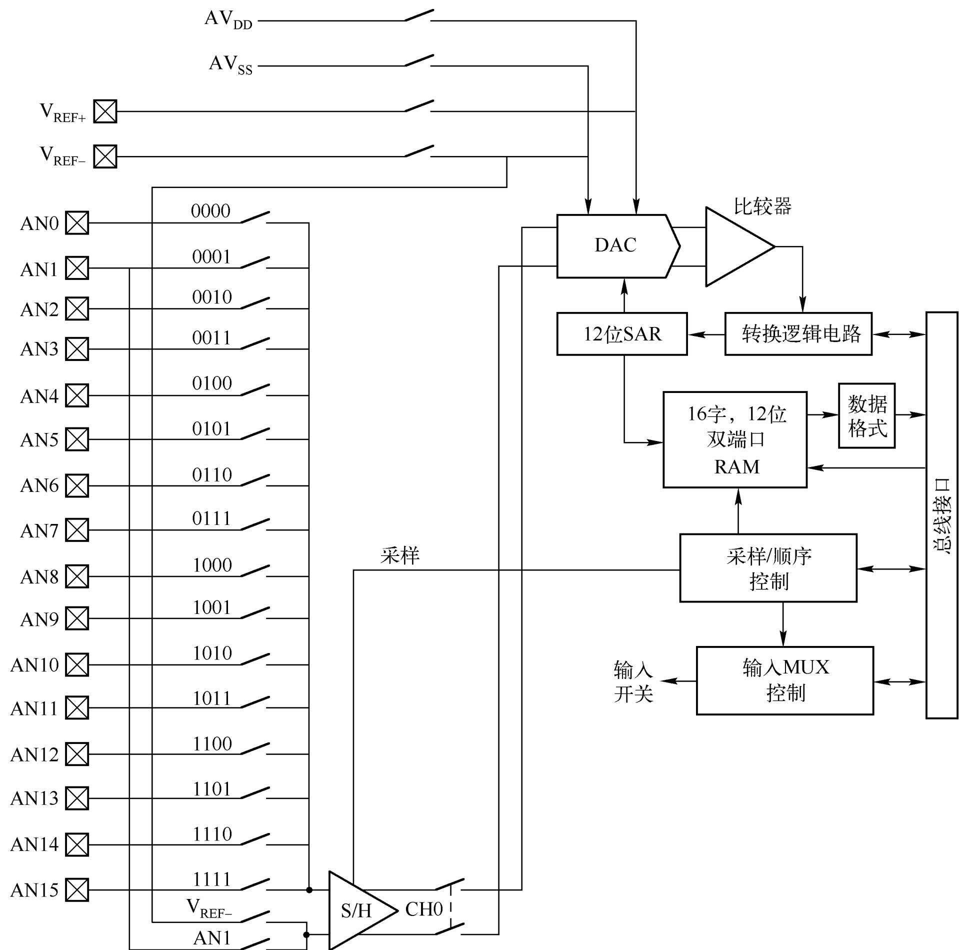


图 1−7　A/D 转换器内部结构

1.3.2　通用定时模块

dsPIC33F 最多可支持 9 个 16 位定时器，即 Timer1～Timer9，其中 8 个 16 位定时器可组合配置成 4 个 32 位定时器，即 Timer2/3、Timer4/5、Timer6/7 和 Timer8/9。定时器一般表现为计数累计功能，通常由时钟脉冲驱动。该时钟可以是 DCS 片内的工作时钟，即内部时

钟，称之为定时器，也可以是由外部引脚输入的时钟，即外部时钟，称之为计数器。无论哪种时钟，定时器的累计都是靠时钟来触发的，触发方式有：下降沿触发、上升沿触发或者两个边沿都触发，这取决于定时器的设置。而累计的方式可以是递增方式、递减方式。另外，定时器还有一个累计上限，当计数达到上限时就会发生溢出。

dsPIC33F 的通用定时器为输入捕捉模块和输出比较/PWM 模块提供基准时钟。定时器具有多种可选模式，既可配置为实时时钟操作，也可以被配置为各种定时器/ 计数器模式。在定时器模式下，定时器模块计数内部时基脉冲；在计数器模式下，模块计数出现在计数器时钟引脚上的外部脉冲数。

对定时器的操作和控制寄存器有：

- TMRx：16 位定时器计数寄存器。
- PRx：与该定时器相关的 16 位周期寄存器。
- TxCON：与该定时器相关的 16 位控制寄存器。

根据某些功能上的差异，定时器分为如下三种类型：

- A 类定时器。
- B 类定时器。
- C 类定时器。

通常 Time1 是 A 类定时器，除了可配置为 16 位的内部定时器/计数器外，Time1 还是一个异步实时时钟计时器。当 Timer1 工作在实时时钟模式（Real Time Clock，RTC）下时，可提供标准日期和时间标记功能，此时 Timer1 使用 32kHz 振荡器作为时钟源，在 CPU 处于空闲或休眠的低功耗状态下运行。A 类定时器的内部结构如图 1-8 所示。

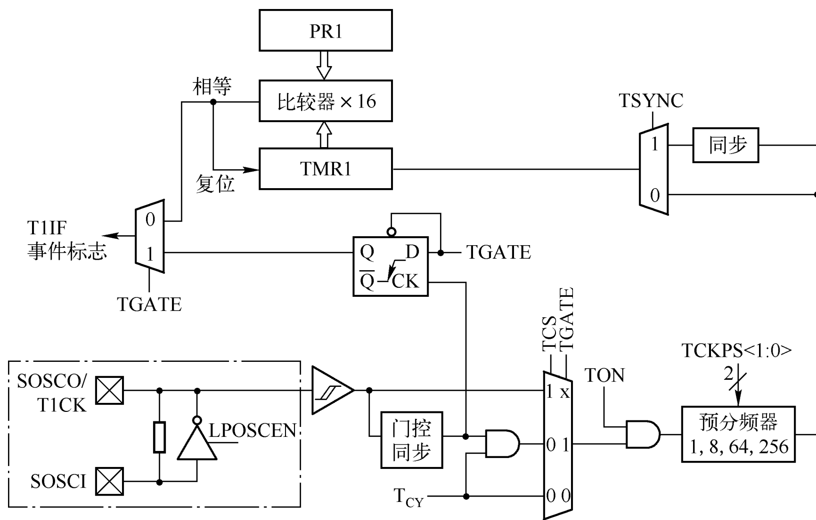


图 1-8　A 类型定时器的内部结构

B 类定时器包括 Timer2、Timer4、Timer6、Timer8，B 类定时器可以和 C 类定时器相连组成 32 位定时器。B 类定时器的 TxCON 寄存器有 32 个控制位，用来使能 32 位定时器功能。B 类定时器的内部结构如图 1-9 所示。

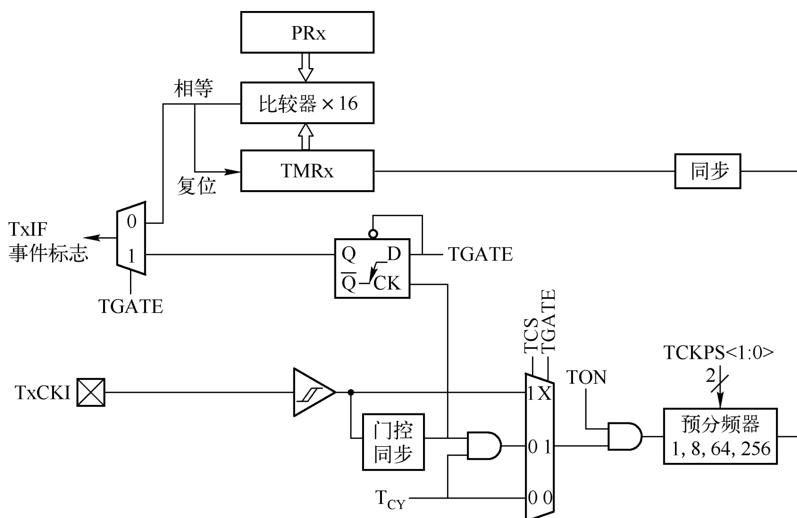


图 1-9　B 类定时器的内部结构

C 类定时器包括 Timer3、Timer5、Timer7、Timer9。Timer2/3、Timer4/5、Timer6/7 和 Timer8/9 可以组成可选的 32 位定时器（可配置成两个 16 位定时器），用于其他外设模块，如输入捕捉模块、输出比较/简单 PWM 模块等，也可以设置为 32 位同步计数器。定时器可以在空闲和睡眠模式下正常运行，当计数值与 32 位周期寄存器的内容相同时产生中断。其中 Timer2/3 还可作为 A/D 转换器的启动触发器。C 类定时器的内部结构如图 1-10 所示。

图 1-10　C 类定时器的内部结构

1.3.3　输入捕捉模块

dsPIC33F 最多可支持 8 路输入捕捉通道，通常用符号 IC1～IC8 表示。输入捕捉模块可捕捉外部输入脉冲的上升沿或下降沿，并记录脉冲跳变的时刻，产生相应的中断。此功能能适合于测量引脚输入的周期方波信号的周期、频率和占空比等，也适用于测量引脚输入的非周

期性矩形脉冲信号的宽度、到达时刻或消失时刻等参数。输入捕捉模块通常应用在测量信号频率和脉冲宽度中，当 ICx 引脚上有信号跳变发生时，输入捕捉模块就会自动捕捉选定时基寄存器的 16 位数值。导致发生捕捉的事件分为以下三种情况：

- 简单捕捉事件：当 ICx 引脚上的输入电平出现下降沿或上升沿时，立即捕捉定时器值。
- 每个边沿（上升沿和下降沿）都捕捉定时器值。
- 预分频捕捉事件：当 ICx 引脚上的输入电平每出现 4 个或 16 个上升沿时，捕捉一次定时器值。

每个输入捕捉通道都可以在两个 16 位定时器（Timer2 或 Timer3）之中选择一个作为时基。选定定时器既可以使用内部时钟，也可使用外部时钟。输入捕捉模块的内部结构如图 1-11 所示。

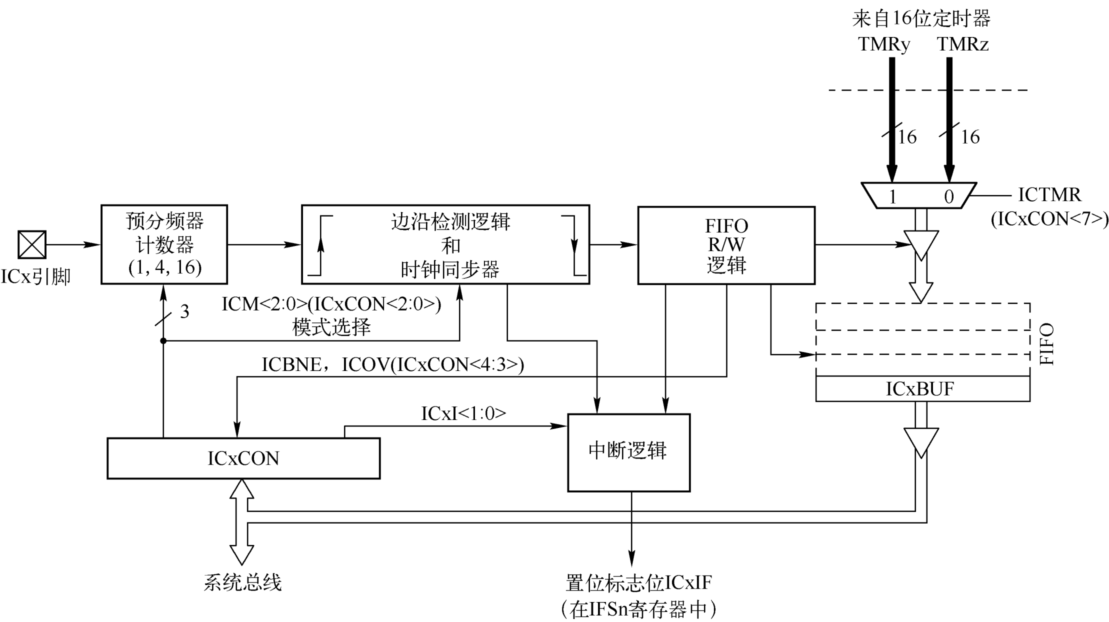


图 1-11　输入捕捉模块的内部结构

当 UART 模块配置为自动波特率工作模式，即 ABAUD = 1(UxMODE<5>)时，输入捕捉模块可以被 UART 模块使用，此时 UART 的 RX 引脚在内部被连接到指定的输入捕捉模块的输入端，与捕捉模块相关的 I/O 引脚将被断开。波特率可以通过在接收到 NULL 字符时测量起始位的宽度来确定。启动自动波特率功能时，捕捉模块必须设置为边沿检测模式（在每个上升或下降沿捕捉）。

1.3.4　输出比较模块

dsPIC33F 最多可支持 8 路输出比较通道，通常用符号 OC1～OC8 表示。输出比较模块通常应用于时序脉冲的控制以及 PWM 调制脉冲的产生中，该模块把选定时基的值与一个或两个比较寄存器的值（取决于选定的工作模式）作比较，在比较匹配事件发生时能产生单个输出脉冲或一连串输出脉冲。输出比较模块的内部结构如图 1-12 所示。

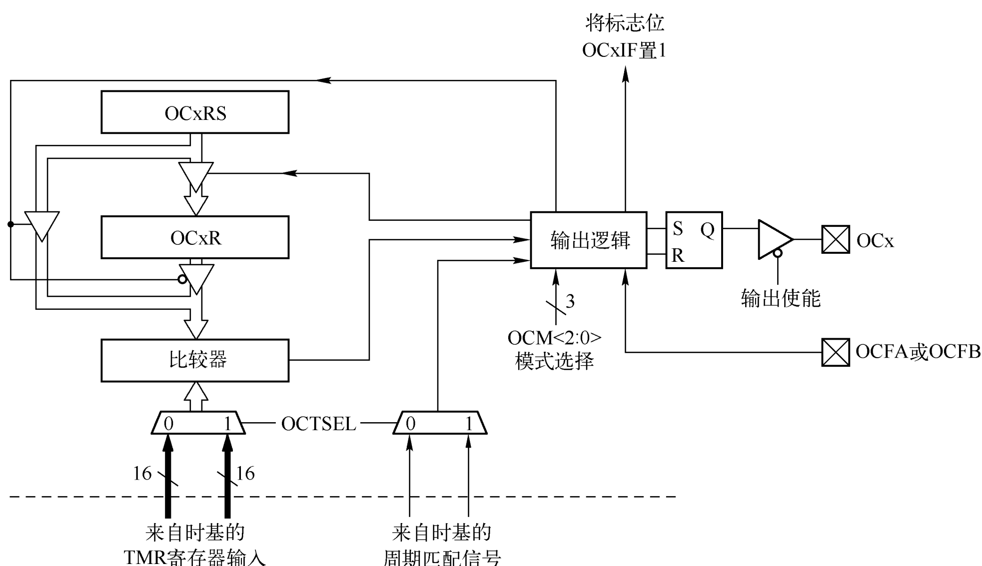


图 1-12　输出比较模块的内部结构

每个输出比较模块都有下列寄存器控制：

- OCxCON：通道的控制寄存器。
- OCxR：输出比较通道的数据寄存器。
- OCxRS：输出比较通道的辅助数据寄存器。

输出比较模块有三种工作模式：

- 单比较匹配模式：把一个值装入 OCxR 寄存器，并将该值与所选的递增计数器寄存器 TMRy 的值作比较，当匹配事件发生时，OCx 引脚会发生跳变。
- 双比较匹配模式：该模式下在处理比较匹配事件时使用 OCxR 和 OCxRS 两个寄存器，先将 OCxR 寄存器的值与递增计数器 TMRy 的计数值作比较，比较匹配事件发生时，在 OCx 引脚上产生脉冲的上升沿；然后 OCxRS 寄存器与同一个递增计数器 TMRy 的计数值作比较，比较匹配事件发生时，在 OCx 引脚上产生脉冲的下降沿。
- 脉宽调制（PWM）模式：在 PWM 模式中，输出比较模块能够输出可变占空比的脉冲波形。OCxR 寄存器保存要输出脉冲的占空比，而 OCxRS 是可由用户写入的缓冲寄存器，以更新 PWM 占空比。在每个定时器与周期寄存器匹配事件产生时（PWM 周期结束时），占空比寄存器 OCxR 就被装载 OCxRS 的内容。

1.3.5　SPI 模块

dsPIC33F 内部集成了两个串行外设接口模块。串行外设接口（Serial Peripheral Interface，SPI）模块是一个同步串行接口，用于与其他外设或者单片机进行通信。这些外设可以是串行 EEPROM、移位寄存器、显示驱动器和 A/D 转换器等。该 SPI 模块可配置为 8 位和 16 位数据收发模式、主从模式以及支持音频编解码的帧同步模式。

SPI 模块由 4 个引脚组成：串行数据输入（SDI）、串行数据输出（SDO）、移位时钟输入或输出（SCK） 和低电平有效从动选择（SS），其中 SS 引脚也可用做帧同步脉冲输入引

脚（FSYNC）。SPI 模块有主从之分，被设置为主器件的 SPI 器件通过 SCK 引脚提供串行通信时钟信号。SPI 模块的主从设备连接示意图如图 1-13 所示。

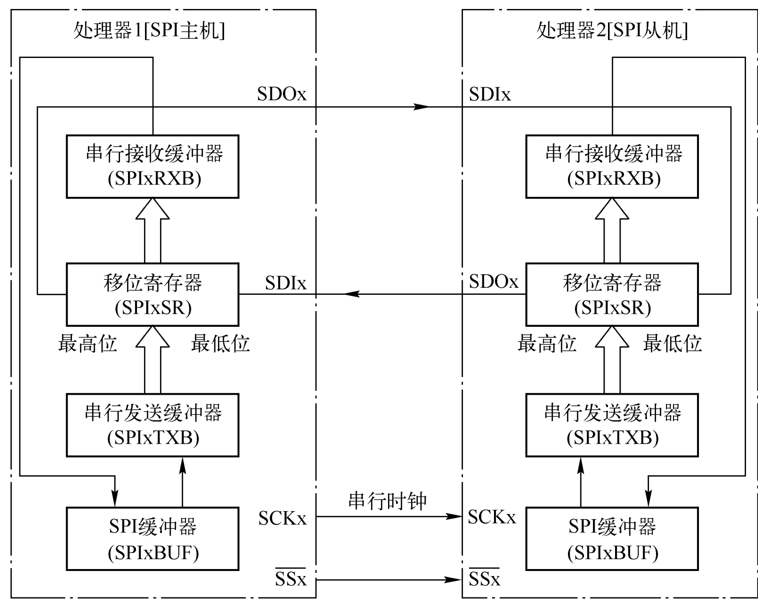


图 1-13　SPI 模块的主从设备连接示意图

SPI 模块包含下列特殊功能寄存器（SFR）。

- SPIxBUF：地址位于 SFR 空间，用于缓冲待发送数据和已接收数据。此地址由 SPIxTXB 和 SPIxRXB 寄存器共享。
- SPIxCON：配置模块各种操作模式的控制寄存器。
- SPIxSTAT：显示各种状态条件的状态寄存器。

此外，还有一个 16 位移位寄存器 SPIxSR，此寄存器不映射到存储器空间，可用于将数据移入和移出 SPI 模块。

存储器映射的 SFR（SPIxBUF）是 SPI 数据接收/发送寄存器。在内部，SPIxBUF 寄存器实际上由两个独立的寄存器（SPIxTXB 和 SPIxRXB）组成。接收缓冲寄存器 SPIxRXB 和发送缓冲寄存器 SPIxTXB 是两个单向 16 位寄存器。这两个寄存器共享名为 SPIxBUF 的 SFR 地址单元。如果用户将需要发送的数据写入了 SPIxBUF 地址单元，该数据会在内部写入 SPIxTXB 寄存器。与此类似，当用户从 SPIxBUF 读取已接收到的数据时，该数据在内部是从 SPIxRXB 寄存器读取的。这种接收和发送操作的双缓冲可以使数据在后台连续传输，发送和接收可同时进行。

1.3.6　UART 接口模块

dsPIC33F 具有一个或多个 UART（Universal Asynchronous Receiver/Transmitter,通用异步接收/发送装置）接口模块，通过该接口 dsPIC33F 可以和其他设备外设通过 RS-232 或 RS-485 接口进行全双工异步通信，UART 使用标准非归零（Non-Return-to-Zero，NRZ）格式（一个启动位，8 或 9 个数据位，一或两个停止位）。硬件提供奇偶校验功

能，用户可以配置成偶校验、奇校验或者不使用奇偶校验、最普通的数据格式是 8 位，没有奇偶校验位，有一个停止位（用 8,N,1 表示）。片上专用的 16 位波特率发生器可用于根据振荡器产生标准的波特率频率。UART 先发送和接收 LSB。UART 的发送器和接收器从功能上讲是独立的，但使用相同的数据格式和波特率。UART 的主要硬件组成是波特率发生器、异步发送器和异步接收器。

波特率发生器包含了一个 16 位的寄存器 UxBRG，控制着 16 位定时器的运行周期。

异步发送器的内部结构如图 1−14 所示，其核心是发送移位寄存器（UxTSR）。该移位寄存器从发送 FIFO 缓冲器 UxTXREG 中获得数据。UxTXREG 寄存器运用软件方式加载数据，直到前一次加载的停止位发送完成后才开始加载 UxTSR 寄存器。停止位发送完毕后，UxTSR 立即从 UxTXREG 寄存器加载新的数据。异步发送器有一个 4 级深、9 位宽的 FIFO 发送数据缓冲器。UxTXREG 寄存器提供对下一个可用的缓冲单元的用户访问。用户最多可在缓冲器中写 4 个字。一旦 UxTXREG 的内容被传送到 UxTSR 寄存器，当前缓冲单元就可以写入新的数据，下一个缓冲单元将成为 UxTSR 寄存器的数据源。缓冲器满了，UTXBF（UxSTA<9>）状态位就会置位。如果用户试图写满缓冲器，则新数据将不会被 FIFO 接收。

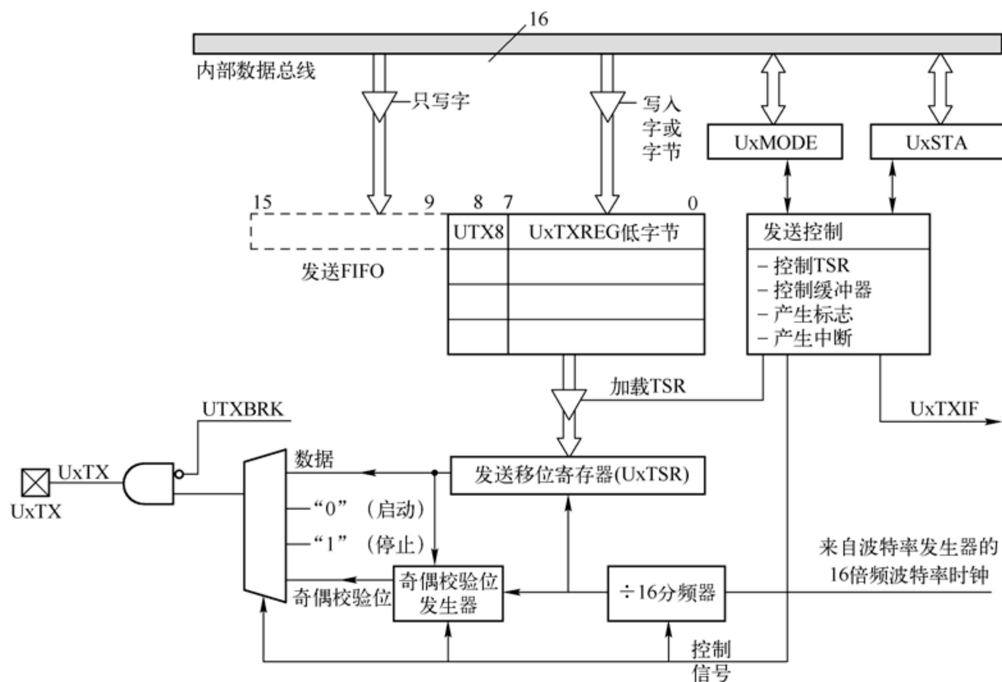


图 1−14　异步发送器的内部结构

异步接收器的内部结构如图 1−15 所示，其核心是接收移位寄存器（UxRSR）。串行异步数据首先在 UxRX 引脚上接收，并送到数据恢复区中。数据恢复区以 16 倍波特率运行，而主接收串行移位器以波特率运行。在采集到 UxRX 引脚上的停止位后，UxRSR 里面的接收到的数据传输到接收 FIFO 中。检测电路通过三次采样 UxRX 引脚上的数据，确定该引脚上是高电平还是低电平。UART 接收器也有一个 4 级深、9 位宽

的 FIFO 接收数据缓冲器。UxRXREG 是一个存储器映射的寄存器，可提供对 FIFO 输出的访问。在缓冲器溢出发生以前，可以有 4 个字的数据被接收并传输到 FIFO 中，从第 5 个字开始将数据移位到 UxRSR 寄存器中。如果 FIFO 已满（4 个字符），而第 5 个字符已经完全接收到了 UxRSR 寄存器，溢出错误位 OERR（UxSTA<1>）将会置位。UxRSR 中的内容将被保留，但是只要 OERR 位被置位，就禁止向接收 FIFO 传输后续数据。用户须在软件中将 OERR 位清零，以允许更多的数据接收。如果需要保存溢出前接收到的数据，用户应该先读所有 5 个字符，然后清零 OERR 位。如果这 5 个字符可以丢弃，用户只要清零 OERR 位即可。这样可以有效地复位接收 FIFO，同时先前接收到的所有数据都将丢失。

图 1-15　异步接收器的内部结构

UART 接口模块具有如下特性：

- 可进行 8 位或 9 位全双工数据传输。
- 可配置为偶校验、奇校验或无奇偶校验方式。
- 具有一个或两个停止位。
- 传输速率从 38bit/s 至 10 Mbit/s 可调，并具有自动波特率检测功能。
- 具有 4 个字符深度的发送数据缓冲器和接收数据缓冲器。
- 当检测到串行数据的启动位后，可自动从休眠模式或空闲模式唤醒。
- 具有相互独立的发送和接收中断，通过配置可实现发送 1 个或 4 个字符产生发送中断，接收 1、3 和 4 字符产生接收中断。
- 支持带地址检测的 9 位模式。

● 支持用于诊断支持的环回模式。

1.3.7 I2C 模块

dsPIC33F 的 I2C 模块是一个用来与其他外设或单片机通信的同步串行接口。这些外设可以是串行 EEPROM、移位寄存器、显示驱动器和 A/D 转换器等。

I2C 总线是一个双线串行接口，该接口有主从之分，在通信时一个器件作为主器件启动总线上的传输并产生时钟信号来允许传输，而其他器件作为响应传输的从器件。时钟线“SCL”是从主器件输出并输入到从器件，数据线“SDA”是双向的，可以是主器件和从器件两者的输出和输入。在 I2C 接口协议中，每个器件都有一个地址。当主器件要开始数据传输时，它首先发送想要“通话”的目标器件地址；所有从器件都会“接听”地址，以确定是否与自身地址匹配。I2C 与 EEPROM 器件的典型接口连接如图 1−16 所示。

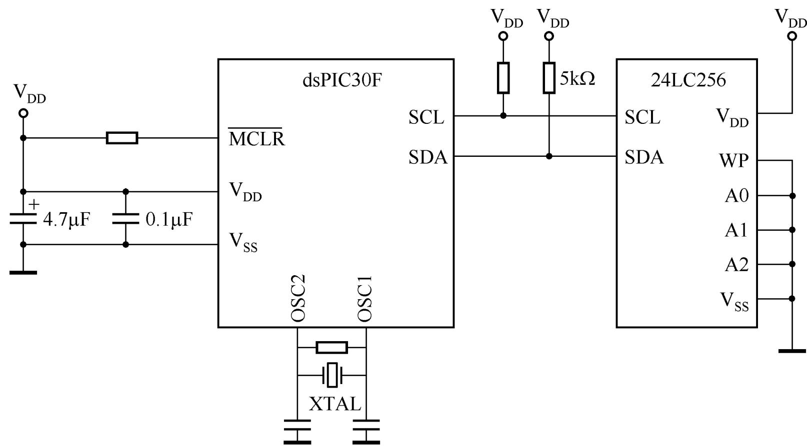


图 1−16 典型的 I2C 接口连接图

I2C 总线上主从设备之间的数据要遵循报文协议，主器件负责控制协议及其时序，从器件要在相应的时序中进行响应。I2C 总线的读取 EEPROM 时序图如图 1−17 所示。

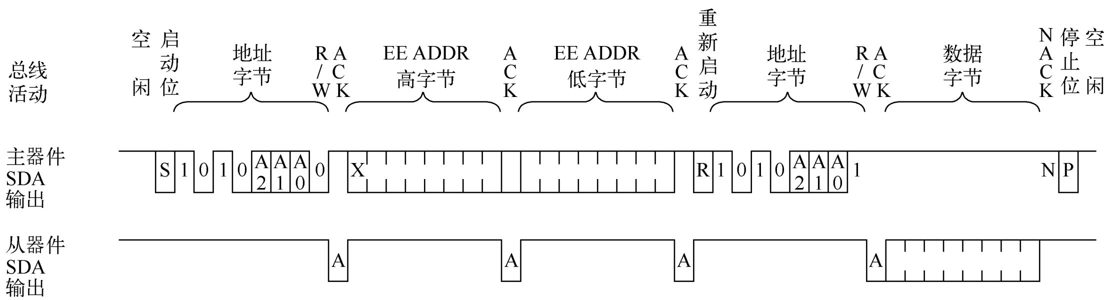


图 1−17 I2C 总线读取 EEPROM 时序图

dsPIC33F 的 I2C 模块支持具有主从的多机工作模式，其功能包括：

- I2C 从动工作支持 7 位和 10 位寻址模式。
- I2C 主控工作支持 7 位和 10 位寻址模式。
- I2C 端口允许主器件和从器件间的双向传输。
- 2C 端口的串行时钟同步能用做握手机制，暂停和恢复串行传输（串行时钟延长）。
- I2C 支持多主机工作模式，能检测到总线冲突并进行相应的仲裁。
- 信号变化率控制，以符合 100kHz 到 400kHz 的总线速率规范。

1.3.8 I/O 引脚

dsPIC33F 提供数量较多的通用 I/O 引脚，一些 I/O 引脚与 dsPIC33F 器件外设功能复用。通常当 dsPIC33F 的相应外设使能时，其对应的引脚将不再作为普通 I/O 引脚使用。所有 I/O 端口引脚都有三个与其工作直接相关的寄存器相对应。数据方向寄存器决定引脚是作为输入引脚还是作为输出引脚；端口数据锁存寄存器提供相应 I/O 引脚的锁存输出数据；端口寄存器提供 I/O 引脚的逻辑电平状态。读端口寄存器获取 I/O 引脚的逻辑电平状态，而写端口寄存器则是将数据值写入端口数据锁存器。I/O 端口引脚具有锁存位（端口锁存寄存器），当读该寄存器时获取锁存的 I/O 内容，当而写该寄存器时则修改锁存的 I/O 的内容，因此如果相应的数据方向寄存器位被配置为输出，则会修改将在引脚上输出的值。这可用于读—修改—写指令中，该指令允许用户修改锁存寄存器的内容，而不受相应引脚状态的影响。

I/O 引脚还具有如下特性：

- 施密特触发器输入。
- CMOS 输出驱动器。
- 内部弱上拉功能。

作为输入时，I/O 引脚能承受 5V 的电信号，并能够兼容不同电压的外部信号。但作为输出时，I/O 引脚只能产生最高为 3.6V 的电平信号。I/O 引脚的输入电平变化可以产生中断请求，即使在时钟被禁止的休眠模式下也可发生中断。可产生中断的 I/O 口共有 24 个（CN0～CN23），每个 CN 引脚还可通过配置实现弱上拉功能。

1.4 dsPIC33F 系列数字信号控制器构成的最小系统

构成以 dsPIC33F 数字信号控制器为核心的最小系统通常包含以下部分：

- 时钟和振荡器控制电路。
- 复位电路。
- 看门狗定时器电路。
- 低功耗电源管理电路。

1.4.1 时钟振荡器控制电路

稳定、准确的时钟信号是 dsPIC33F 系统工作的必要条件。dsPIC33F 提供多种可选的时钟振荡器控制电路以适应不同环境的应用。dsPIC33F 可使用 4 个时钟源中的一个提供系统时

钟，这些时钟源是主振荡器、辅助振荡器、内部快速 RC（FRC）振荡器和低功耗 RC（LPRC）振荡器。

主振荡器的工作原理如图 1−18 所示，其时钟信号来源于 OSC1 和 OSC2 引脚，根据连接的晶体振荡频率的不同，分成 XT、XTL 和 HS 三种模式：

- XT（晶振）：3～10MHz 范围内的晶振和陶瓷谐振器，晶振连接在 OSC1 和 OSC2 引脚之间。
- HS（高速晶振）：10～40MHz 范围内的晶振，晶振连接在 OSC1 和 OSC2 引脚之间。
- EC（外部时钟）：0.8～64MHz 范围内的外部时钟信号，外部时钟信号直接连接到 OSC1 引脚。

辅助振荡器（LP）是为低功耗运行而设计的，使用 32kHz 晶振或陶瓷谐振器。LP 振荡器使用 SOSCI 引脚和 SOSCO 引脚。

FRC 振荡器是快速（标称值为 8MHz）的内部 RC 振荡器。该振荡器旨在不使用外部晶振、陶瓷谐振器或 RC 网络的情况下使系统能正常工作。

LPRC（低功耗 RC）振荡器是对定时器要求不是很高的应用场合采用低成本 RC 的振荡器方式。RC 振荡器的频率取决于振荡电阻、电容、工作电压以及环境温度等。LPRC 振荡器的工作原理如图 1−19 所示。

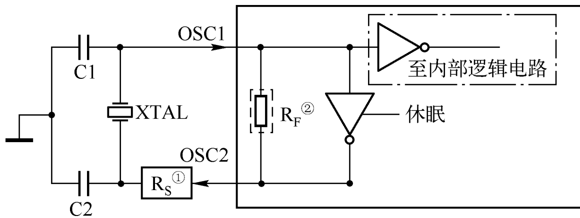


图 1−18　主振荡器的工作原理

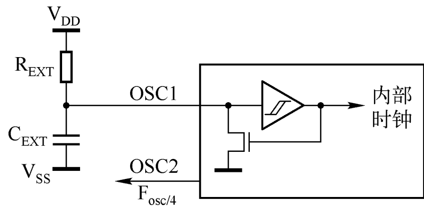


图 1−19　LPRC 振荡器的工作原理

① 对于AT条形切割的晶体可能需要串联一个电阻R_S。

② 内部反馈电阻R_F，通常范围为 2~10MΩ。

1.4.2　复位电路

dsPIC33F 器件内部集成了复位电路，可以不需要外部扩展，复位系统的内部结构如图 1−20 所示。dsPIC33F 的复位源有多个，包括上电复位（POR）、欠电压复位（BOR）、看门狗定时器复位（WDTR）、引脚复位（EXTR）、RESET 指令以及出错复位等，最后通过一个内部系统复位信号（SYSRST）启动 dsPIC33F 器件复位。

当给 dsPIC33F 器件加电时会产生上电复位，如果供电电压降至器件门限电压（VPOR）以下时，则将产生一个新的上电复位事件。上电复位会产生一个内部 POR 脉冲，该脉冲会激活芯片内部的主复位信号（SYSRST）

欠电压复位（BOR）模块依靠内部参考电压电路在发生欠电压条件时产生器件复位。欠电压条件通常由电源线故障（即由于不良的电源传输线路产生的交流信号波形部分丢失或由于当接入过大的电感负载时电流过大而导致电压下降）产生。通过配置，可选择下面电压跳变点中的一个作为复位条件：2.0V、2.7V、4.2V 和 4.5V。

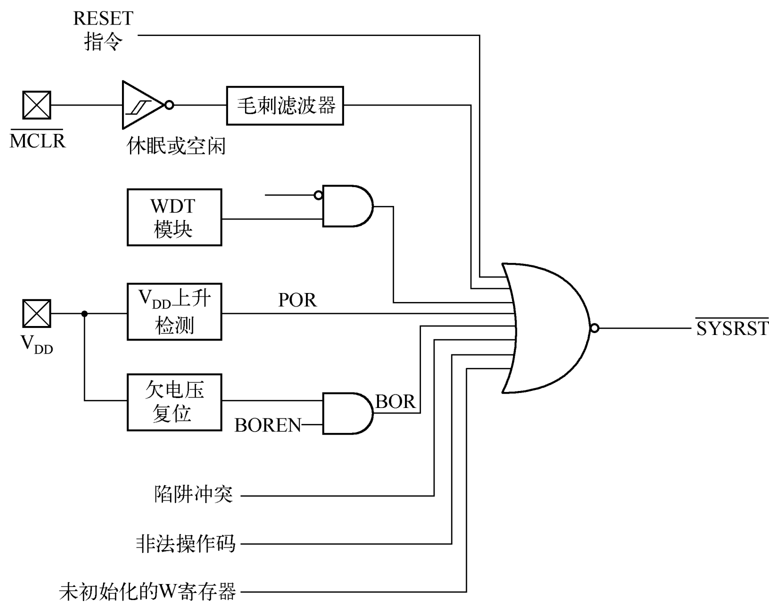


图 1−20　复位系统的内部结构

1.4.3　看门狗定时器电路

看门狗定时器电路（WDT）的主要功能是在出现软件异常事件时复位处理器。WDT 是自由运行的定时器，它在 LPRC 振荡器上运行而不需要外部组件。因此，即使系统时钟源（如晶体振荡器）出现故障，WDT 定时器仍然会继续工作。

WDT 的使能或禁止由 FWDT 器件配置寄存器中的 FWDTEN 器件配置位控制。FWDT 寄存器的值在器件编程时被写入。当 FWDTEN 配置位置 1 时，WDT 被使能。这是一个已被擦除的器件的默认值。当 FWDTEN 配置位被编程为“0”后，用户可以软件控制 WDT。通过置位 SWDTEN 控制位（RCON<5>）用软件使能 WDT。在任何器件复位时 SWDTEN 控制位都会被清零。

如果看门狗定时器被使能，WDT 将进行加计数直到溢出或“超时”。除了在休眠或空闲模式，WDT 超时将强制器件复位。要阻止 WDT 超时复位，用户必须使用 CLRWDT 指令定期清零看门狗定时器。CLRWDT 指令也将清零 WDT 预分频器。如果 WDT 在休眠或空闲模式超时，器件将唤醒并从 PWRSAV 指令处开始继续执行代码。

看门狗定时器的内部结构如图 1−21 所示。

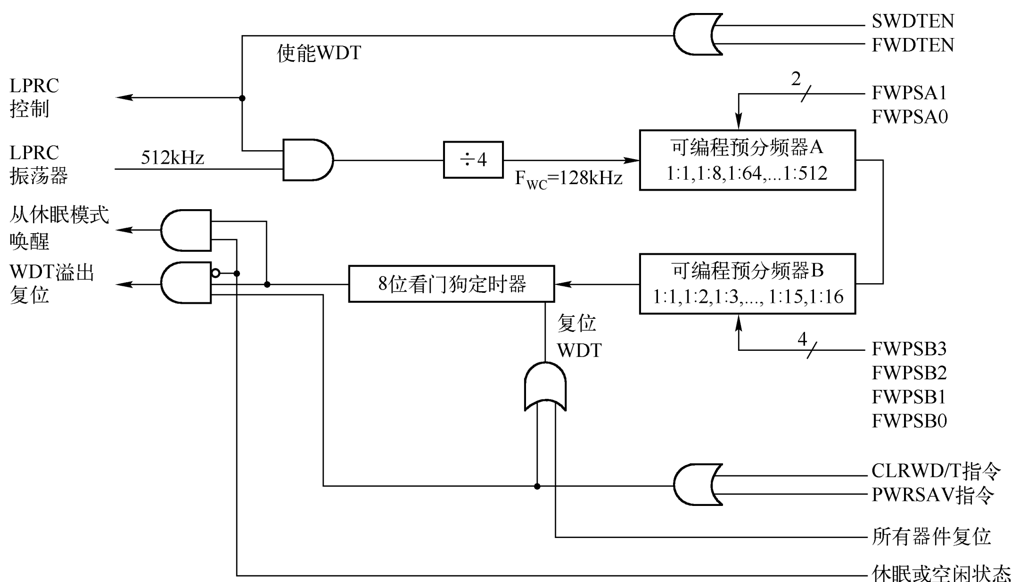


图 1-21　看门狗定时器的内部结构

1.4.4　低功耗电源管理电路

dsPIC33F 器件可以通过执行 PWRSAV 指令进入两种低功耗模式：

- 休眠模式：CPU、系统时钟源和任何依靠系统时钟源工作的外设都被禁止。这是器件的最低功耗模式。
- 空闲模式：CPU 被禁止，但是系统时钟源继续工作；外设继续工作，但可以有选择地禁止。

休眠模式具有如下特性：

- 系统时钟源关闭，如果使用了片上振荡器，则其也将被关闭。
- 在没有 I/O 引脚输出电流的前提下，器件当前功耗最小。
- 由于系统时钟源被禁止，所以故障保护时钟监视器（FSCM）在休眠模式下不工作。
- 如果 WDT 被使能，LPRC 时钟将继续在休眠模式下运行。
- 低压检测电路如果被使能，则继续在休眠模式下保持工作。
- BOR 电路如果被使能，则继续在休眠模式下工作。
- WDT 如果被使能，则在进入休眠模式之前自动清零。

某些外设可能会继续在休眠模式运行，这些外设包括检测输入信号电平变化的 I/O 引脚，或使用外部时钟输入的外设。任何根据系统时钟源工作的外设都会在休眠模式禁止。

发生下列事件之一时，处理器将从休眠模式退出或被唤醒：

- 任何单独允许的中断源。
- 任何形式的器件复位。
- WDT 超时。

当器件进入空闲模式时，将发生以下事件：

- CPU 将停止执行指令。

- WDT 被自动清零。
- 系统时钟源将保持有效，而且在默认情况下外设模块将使用系统时钟源继续正常工作。
- 如果 WDT 或 FSCM 被使能， LPRC 也将保持有效。

在发生以下事件时，处理器将从空闲模式唤醒：

- 任何单独允许的中断。
- 任何器件复位源。
- WDT 超时。

CPU 优先级分配为零的用户中断源不能将 CPU 从空闲模式唤醒，因为此中断源被有效禁止了。要使用中断作为唤醒源，此中断的 CPU 优先级必须被分配为 1 或更高。在从空闲模式唤醒时，时钟再次供 CPU 使用且指令立即从 PWRSAV 指令之后的一条指令或 ISR 中的第一条指令开始执行。

1.5　无线通信中的数字信号处理技术

随着人们对通信多样性需求的增长，近年来无线通信得到长足发展，尤其是近距离、低成本、高速率的通信技术更是发展迅猛，例如无线局域网、蓝牙技术以及无线传感网络等。无线通信是指信号传输的信道是无线信道，人们通常根据无线电波的频率把无线信道分为长波、中波、短波、超短波、微波等信道，不同波长的无线电波其传输特性不同。信号在无线信道中传输有以下两种基本形式：

1）模拟信号。其信号的波形可以表示为时间的连续函数。这里，“模拟”两字的含义是指用电参量（如电压、电流）的变化来模拟信息源发送的消息。如电话信号就是话音声波的电模拟，它是利用送话器的声/电变换功能，把话音声波压力的强弱变化转变成话音电流的大小变化。以模拟信号为传输对象的传输方式称为模拟传输，而以模拟信号来传送消息的通信方式称为模拟通信。

2）数字信号。数字信号的特征是其幅度不随时间连续变化，只能取有限个离散值。通常以取两个离散值（“0”和“1”）来表示二进制数字信号，由于传统的模拟信号在传输时存在抗干扰性能差、保密性低、不便于存储等缺点，目前的无线通信普遍采用数字通信技术。典型的无线数字通信系统的模型如图 1−22 所示。

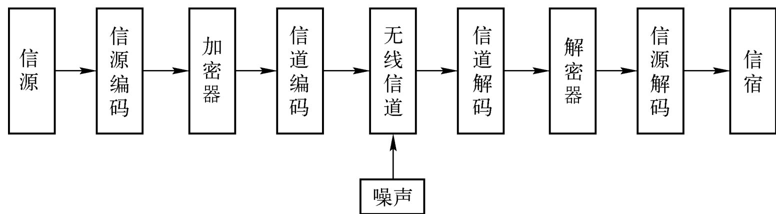


图 1−22　典型的无线数字通信系统的模型

图中信源负责把信息变换成原始电信号，例如声音、图像以及其他传感器所提供的信息。

信源编码的任务是把模拟信号变换成数字信号，以完成模拟/数字转换（A/D 转换），对于原来已经是数字信号的信源则可省略信源编码。

加密器的功能是对数字信号进行加密处理，保证传输信息的安全性。

信道编码通常由纠错编码和线路编码两部分组成。由于无线信道存在噪声，信息在传输过程中会产生差错，为了使接收端能够自动检测出错码或者纠正错码，必须进行纠错编码。通常的做法是增加一些冗余码元，使码元之间形成较强的规律性。线路编码是指把数字信号转变成适合于信道上传输的波形。

为了适应无线信道传输的频带要求，需要在信道编码后将数字信号频谱搬移到无线信道的高频范围内，这一变化称为调制。未经调制的数字信号称为基带信号，调制后的信号称为频带信号。

无线信道是传输无线数字信号的媒介。无线信号通过天线发射出去，采用电磁波的方式在空间传播。在传输过程中不可避免地受到系统内部和外部的干扰，这里把所有的干扰（包括内部噪声）都折合为一个统一的等效噪声源来表示。

接收端的解调、信道解码、解密器、信源解码等功能与发送端的调制、信道编码、加密器和信源编码等功能是一一对应的反交换。

在无线数字通信系统中人们经常用到数字处理技术，常用的数字处理技术包括数字滤波器、数字调制技术、同步控制技术以及差错控制技术等。

1.5.1 数字滤波器

数字滤波器在数字信号处理中有着广泛的应用，其作用是将输入信号中的有用频率分量输出，并阻止无用的频率分量通过。从结构上看，数字滤波器是由数字乘法器、加法器和延时单元组成的一种装置。其功能是对输入离散信号的数字代码进行运算处理，以达到改变信号频谱的目的。由于电子计算机技术和大规模集成电路的发展，数字滤波器已可用计算机软件实现，也可用大规模集成数字电路硬件实时实现。

数字滤波器是一个离散时间系统（按预定的算法，将输入离散时间信号转换为所要求的输出离散时间信号的特定功能装置）。应用数字滤波器处理模拟信号时，首先须对输入模拟信号进行限带、抽样和模数转换。数字滤波器输入信号的抽样率应大于被处理信号带宽的两倍，其频率响应具有以抽样频率为间隔的周期重复特性，且以折叠频率即 1/2 抽样频率点呈镜像对称。为得到模拟信号，数字滤波器处理的输出数字信号须经数模转换、平滑等处理过程。数字滤波器具有高精度、高可靠性、可程控改变特性或复用、便于集成等优点。

数字滤波器有低通、高通、带通、带阻和全通等类型。它可以是时不变的或时变的、因果的或非因果的、线性的或非线性的。应用最广的是线性、时不变数字滤波器。

从数字系统的框图结构或流图结构上分，数字滤波器分为 IIR（无限脉冲响应）和 FIR（有限脉冲响应）两类。

（1）IIR（无限脉冲响应）滤波器

一个 N 阶 IIR 滤波器的传递函数可以表示为

$$H(z)=\frac{\sum_{k=0}^{M}b_k z^{-k}}{1+\sum_{k=1}^{N}a_k z^{-k}} \tag{1-1}$$

IIR 数字滤波器采用递归型结构，即结构上带有反馈回路。其运算结构通常由延时、乘以系数和相加等基本运算组成，可以组合成直接型、正准型、级联型、并联型 4 种结构形式，都具有反馈回路。由于运算中的舍入处理，使误差不断累积，有时会产生微弱的寄生振荡。

IIR 数字滤波器在设计上可以借助成熟的模拟滤波器的成果，如巴特沃斯、切比雪夫和椭圆滤波器等，有现成的设计数据或图表可查，其设计工作量比较小，对计算工具的要求不高。在设计一个 IIR 数字滤波器时，首先根据指标写出模拟滤波器的公式，然后通过一定的变换，将模拟滤波器的公式转换成数字滤波器的公式。 IIR 数字滤波器的缺点是相位特性不好控制，对相位要求较高时，需加相位校准网络。另外，反馈还使 IIR 滤波器的数字运算可能溢出。

（2）FIR（有限脉冲响应）滤波器

FIR 滤波器的传递函数如下：

$$H(z)=\sum_{k=0}^{K-1}h(k)z^{-k} \tag{1-2}$$

FIR 滤波器最重要的优点就是由于不存在系统极点，所以 FIR 滤波器是绝对稳定的系统。FIR 滤波器还确保了线性相位，这在信号处理中也非常重要。此外，由于不需要反馈，FIR 滤波器的实现也比 IIR 滤波器简单。FIR 滤波器的缺点在于它的性能不如同样阶数的 IIR 滤波器，不过由于数字计算硬件的飞速发展，这一点已经不成为问题。再加上引入了计算机辅助设计，FIR 滤波器的设计也得到极大的简化。基于上述原因，FIR 滤波器比 IIR 滤波器的应用更广。

1.5.2　数字调制技术

无线信道通常采用调制技术才能实现传输。“调制”是指通过基带信号对载波波形的某些参量进行控制，使这些参量随基带信号的变化而变化。当然，调制的目的不仅是为了满足信道传输的需要，还在于实现多路复用、完成频率分配和减小噪声干扰的影响等。在无线数字通信中，对受调载波的波形，原理上并无特殊的要求，一般选用正弦信号作为载波，这是因为它的形式简单，便于生成与接收。由于正弦信号的三个参量（幅度、频率和相位）都能携带信息，因而相应地有调幅、调频和调相三种基本调制形式，而且还可以由此派生出多种形式。数字通信一般采用数字调制，它与模拟调制的原理是相同的，两者的区别仅在于：模拟调制是对载波信号的参量进行连续调制，在解调时必须对接收信号的调制参量连续地进行估值；而数字调制是用载波信号参量的离散状态来表征所传输的数据信息，在解调时只需对载波信号的受调参量进行检测和判决。因此数字调制信号也称为键控信号。常用的数字调制方式有振幅键控（Amplitude Shift Keying，ASK）、移频键控（Frequency Shift Keying，FSK）和移相键控（Phase Shift Keying，PSK），图 1−23 给出了这三种信号波形的示例。

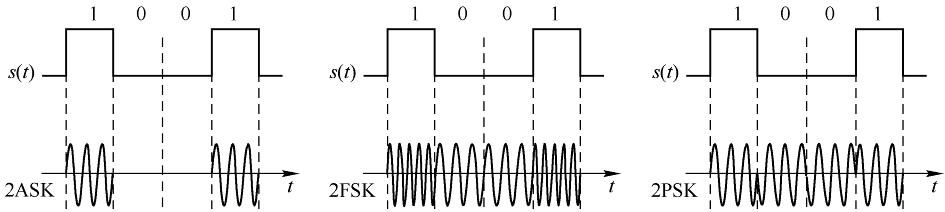


图 1−23　正弦载波的三种键控波形

（1）数字幅度调制

用携带数据信息的矩形基带信号去控制正弦载波的幅度，称为幅度键控（ASK），又称通断键控（OOK）。在生成 2ASK 信号时，传送“1”信号，即输出正弦载波信号 $A\cos(\omega_c t+\varphi_c)$；而传送“0”信号，输出为 0。这相当于用一个单极性矩形基带信号（含有直流分量）与正弦载波信号相乘。所以二进制幅度键控的调制器可以用一个相乘器来实现。图 1−24 表示了 2ASK 信号的生成原理框图及波形。

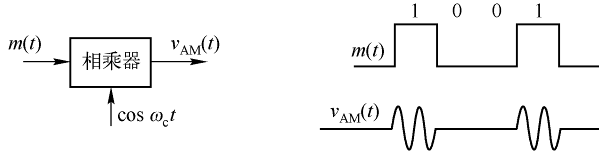


图 1−24　2ASK 信号生成原理框图及波形

假设携带数据信息的基带信号 $m(t)$ 为随机的单极性矩形脉冲序列，即

$$m(t)=\sum_{n=-\infty}^{\infty}a_n g(t-nT_s) \tag{1−3}$$

其中

$$a_n=\begin{cases}0, & \text{以概率 } p \text{ 出现}\\ 1, & \text{以概率} 1-p \text{ 出现}\end{cases}$$

$$g(t)=\begin{cases}1, & 0 \text{剟} t \quad T_s\\ 0, & \text{其他}\end{cases}$$

那么，已调信号 $v_{AM}(t)$ 为

$$\begin{aligned}v_{AM}(t)&=m(t)A\cos(\omega_c t+\varphi_c)\\ &=\left[\sum_{n=-\infty}^{\infty}a_n g(t-nT_s)\right]A\cos(\omega_c t+\varphi_c)\end{aligned} \tag{1−4}$$

对 2ASK 信号的解调主要有两种方法：包络检波法和相干解调法。包络检波法是利用包络检波器对幅度键控信号进行检波以恢复其基带信号。相干解调法由相乘器和低通滤波器组成，输入已调信号与相干载波信号在相乘器相乘后，再由低通滤波器过滤即可得到所需的基带信号。

（2）数字频率调制

数字频率调制是利用基带数字信号控制载波频率的变化来传输数字信息的一种调制形式。最简单的数字频率调制是二进制频移键控（2FSK）。由于它的抗噪声、抗衰落性能优于 ASK，设备又不复杂，实现也较容易，所以一直被广泛应用在中、低速数据通信系统中。但是，由于在功率和频率利用率方面，传统的 2FSK 不及相移键控（PSK），所以在差分相移键控（DPSK）取得成功之后，就逐渐被取而代之。然而，2FSK 可视为两路不同载频的 ASK 复合信号，具有潜在的二重频率分集作用，因此在一些衰落信道（如短波、散射信道）的传

输中，仍得到了应用。近年来，数字频率调制技术有了很大的进步，特别是多进制频移键控（MFSK）、连续相位频移键控（CP-FSK）、最小频移键控（MSK），以及高斯最小频移键控（GMSK）和软化调频（TFM）等技术的发展，使得其在数字卫星通信系统和无线电信道中也得到了应用。

图 1−25 给出了利用频率选择法生成 2FSK 信号的原理框图及波形。在某一码元期间，当输入“1”时输出频率为 $f_1(t)$ 为信号；反之，当输入“0”时，输出频率为 $f_2(t)$ 的信号。

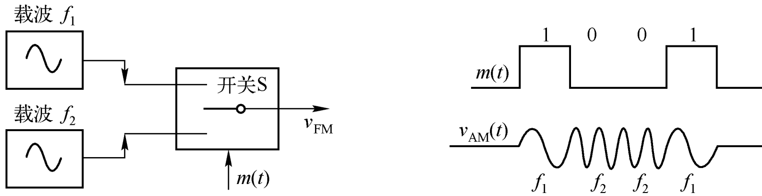


图 1−25　2FSK 信号生成原理及波形

若设携带数据信息的基带信号 $m_1(t)$ 为随机的单极性矩形脉冲序列。在每一个码元期间，载频为 $f_1(t)$ 和 $f_2(t)$ 的正弦信号分别用来传输二进制数字“1”和“0”。此时，载频为 $f_1(t)$ 的幅度键控信号的时域表达式为

$$\begin{aligned} S_1(t) &= m_1(t)A\cos(\omega_1 t+\varphi_1) \\ &= \sum_{n=-\infty}^{\infty} a_n g(t-nT_{\text{s}})A\cos(\omega_1 t+\varphi_1) \end{aligned} \tag{1−5}$$

式中，T_{s} 为码元宽度；a_n 和 $g(t)$ 分别满足如下关系：

$$a_n=\begin{cases} 0, & \text{以概率 } p \text{ 出现} \\ 1, & \text{以概率 } 1-p \text{ 出现} \end{cases}$$

$$g(t)=\begin{cases} 1, & 0\text{剟}t \quad T_{\text{s}} \\ 0, & \text{其他} \end{cases}$$

由于在任一码元期间，载频 $f_2(t)$ 是受 $\{a_n\}$ 的反码序列 $\{\overline{a_n}\}$ 控制的，因此它的幅度键控信号可表示为

$$\begin{aligned} S_2(t) &= m_2(t)A\cos(\omega_2 t+\varphi_2) \\ &= \sum_{n=-\infty}^{\infty} a_n g(t-nT_{\text{s}})A\cos(\omega_2 t+\varphi_2) \end{aligned} \tag{1−6}$$

于是，由载频为 $f_1(t)$ 和 $f_2(t)$ 两个幅度键控信号合成的 2FSK 信号可表示为

$$\begin{aligned} S(t) &= S_1(t)+S_2(t) \\ &= \sum_{n=-\infty}^{\infty} a_n g(t-nT_{\text{s}})A\cos(\omega_1 t+\varphi_1)+ \\ &\quad \sum_{n=-\infty}^{\infty} \overline{a_n} g(t-nT_{\text{s}})A\cos(\omega_2 t+\varphi_2) \end{aligned} \tag{1−7}$$

对 2FSK 信号的解调有相干和非相干解调两种方法。这两种方法各自还可派生出若干种

具体方法。究竟选用何种解调方法，应根据发送 FSK 信号的形式及参数、对解调器抗干扰的要求、解调技术的可实现性和设备成本等因素综合考虑。目前，常用的是非相干解调法，虽然它的抗干扰性能不及相干解调法优越，但解调时无需从 FSK 信号中提取相干载波，因而实现起来比较简单。

（3）数字相位调制

数字相位调制是利用基带数字信号控制载波相位的变化来传输数字信息的一种调制形式。假设载波的相位是对一个固定参考相位而言的，以数字基带信号 $m(t)$ 的“1”对应于已调信号 $S(t)$ 中载波的 0 相位，而以 $m(t)$ 的“0”对应于 $S(t)$ 是载波的 π 相位（此假设也可反过来）。由于在这种调制过程中，已调信号的相位变化是相对于一个固定的参考相位而言的，所以称之为“绝对移相”。其信号波形如图 1−26 所示。

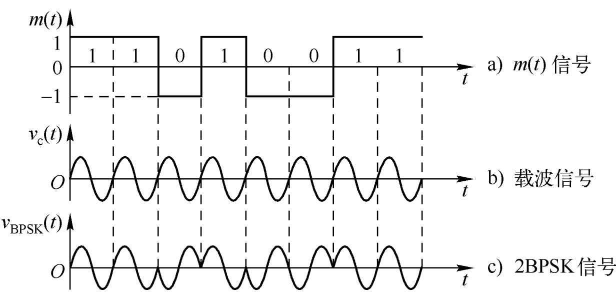


图 1−26　2PSK 信号波形

2PSK 信号的时域表达式为

$$S(t)=\sum_{n=-\infty}^{\infty} g(t-nT_s)\cos(\omega_c t+\varphi_n) \tag{1-8}$$

式中，ω_c 为载波角频率；φ_n 是第 n 个码元的载波相位；T_s 为码元持续时间；$g(t)$ 为数字基带信号波形；φ_n 是一随机变量，它只有两种可能的取值：0 或 π 。

2PSK 信号的生成方法有两种：调相法和相位选择法。用相位选择法产生 2PSK 信号时，应预先把所需相位的载波准备好，以便由数字基带信号来选择相应的载波输出。振荡器和反相器分别输出 0 和 π 两种不同相位的载波，输入基带信号为单脉冲序列。该电路的输出是载波相位随数字基带信号变化的调相信号。图 1−27 是 2PSK 的调制与解调框图。

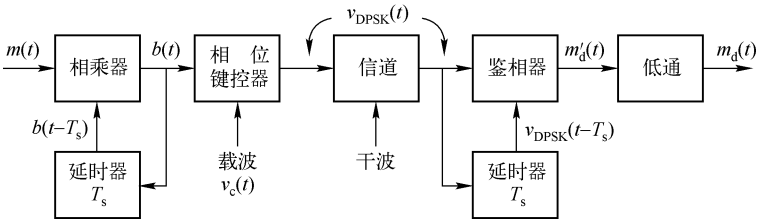


图 1−27　2PSK 的调制与解调框图

在 2PSK 信号的解调系统中，同步载波恢复会有 180° 的相位模糊问题，对 2PSK 系统误码性能影响很大，所以 2PSK 方式在实际中很少采用。为了解决 2PSK 信号解调过程的反

向工作问题，提出了二进制差分相位键控（2DPSK）。2DPSK 方式是利用前后相邻码元的载波相位的相对变化来表示数字信息的。假设前后相邻码元的载波相位差为$\Delta\varphi$（$\Delta\varphi$定义为本码元初相与前一码元初相之差），并设数字信息与$\Delta\varphi$之间的关系为

$$\Delta\varphi=\begin{cases}0, & \text{表示数字信息“0”} \\ \pi, & \text{表示数字信息“1”}\end{cases}$$

2DPSK 信号的解调方法有两种：极性比较法和相位比较法。极性比较法就是相干解调法，此方法所需的相干载波是从接收信号中提取的。因为相干解调后仍是相对码，所以最后还需经码变换器将相对码变换成绝对码。输入的 2DPSK 信号在相乘器中与相干载波相乘，然后经低通滤波（用积分器亦可）。当接收信号与载波同相时，滤波器输出正脉冲；否则，输出负脉冲。然后经取样判决和码元形成，便可得到相对码输出。最后通过码变换器，还原成原信息码。

1.5.3 同步控制技术

无线数字通信中同步是实现通信的前提。数据通信系统能够有效、可靠地工作，在很大程度上取决于性能良好的同步系统。按照要求同步的对象不同，数据通信系统的同步包括：载波同步、位同步、群同步及网同步。

1．载波同步

在采用频带传输的相干解调系统中，接收端必须提供一个与发送端同频同相的相干载波，这一过程称为载波同步。对载波同步的基本要求是：同步误差（确切地说是相位误差）小；建立同步的时间短；同步保持时间长；为同步所占用的功率小及频带窄。实现载波同步的方法可分为两类：如果接收的已调信号频谱中已含有载波分量或者载波导频分量，则可用带通滤波器或锁相环直接提取；而对于抑制载波而又没有插入导频的已调信号，则通过对其进行非线性变换或采用特殊的锁相环来获取相干载波。

常用的载波同步方法有：

（1）插入导频法

插入导频法又称外同步法，发送端在发送有用信号频谱的同时在适当的位置再加入一个低功率的线谱（其对应的正弦信号称为导频信号），这样接收端就可以利用窄带滤波器把它提取出来，再经适当处理后形成相干载波。导频的频率应与载频有关或者就是载频。

（2）直接提取法

直接提取法又称自同步法。它适用于接收信号中含有载波分量或者对接收信号（如 PSK 信号）进行某种非线性变换后，含有载波的谐波分量的场合。常用的方法有平方环法和同相正交法（科斯塔斯环）

2．位同步

无线数字通信中，数据信号是以码元形式逐个地发送和接收的，这要求发、收双方的时钟要有一个稳定而可靠的同步关系。另外，在接收端无论是经解调得到的基带信号，还是由基带传输而直接得到的基带信号都可能存在一定程度的畸变和干扰，这也要求本地码元定时与发送端定时脉冲的频率相同，且选择最佳的判决时刻以保证对接收信号的最佳取样和判决。上述要求都是通过位同步的作用来得到满足的。通常，把在接收端产生码元定时信号的过程，称为位同步或码元同步。

常用的位同步生成方法有：

（1）插入导频法

为了获取码元定时信号，必须先确定接收到的基带信号中是否存在位定时的频率分量。如果存在此频率分量，就可用滤波器直接把位定时信息提取出来。而对某些本身不包含位定时信息的基带信号（如随机的二进制不归零码），则有必要在基带信号中插入位同步的导频信号，或者对该基带信号进行某种码型变换，以达到获取位定时信息的目的。

（2）直接提取法

直接提取法是由接收端直接从接收信号中提取位定时信息。对于不包含位定时频率分量的基带信号，必须对其进行波形变换，使变换后的基带信号中含有离散的位定时分量，这样就可用窄带滤波器或锁相环来提取所需的位定时信息，继而形成位同步信号。直接提取法有滤波法和锁相法两种。

3．群同步

无线数字通信中通常将传输的码元序列按照一定的要求（以字符、字符组、报文等形式）进行分组，这些分组称为帧或信息包。为了确保这些分组的正确接收，在传输时应产生与分组保持同步的定时信号。这种实现帧或信息包同步传输的过程称为群同步。

对通信过程中的一个分组而言，分组内数据信息的任意两个码元之间的距离都是单元码元长度的整数倍，收、发之间必须保持位同步，才能实现可靠的通信。这实际上是一种同步传输方式的通信，而位同步就是实现同步通信方式的手段。但是分组的长度是有限的，不同分组之间的时间间隔也不一定是单个码元时间的整数倍，多个分组或多个群之间的通信采用的是异步传输方式，群同步是实现异步传输方式的必要保证。

群同步的任务是对解调器输出的码元序列进行正确的分组。与载波同步、位同步不同，群同步一般是通过数据格式的特殊设计来实现的，即通过在数据码元序列中插入特定的同步码元或同步码组来实现群同步。因此，实现群同步的关键在于如何识别插入的同步标志。

对群同步的基本要求是：

- 同步可靠性高，即漏同步率和假同步率低。
- 同步平均建立时间短。
- 为实现群同步而插入到数据码元序列中的群同步码元或群同步码组的冗余度小。

实现群同步的方法有两类：一类是在发送的码元序列中插入专门设计的群同步码元或群同步码组，这类方法称为外同步法；另一类是利用码元序列的本身特性来提取群同步信号，这类方法称为内同步法。

常用的外同步方法有：

（1）起止位同步法

这是一种利用起止位实现异步传输的方法。被传输的单位是一个字符，并用起始位表示字符的开始，用停止位表示字符的结束。因此，群是由起始位、字符位及停止位构成的。数据终端与计算机之间通过 RS-232 串行接口进行通信就是一个利用起止位建立群同步的例子。在通信过程中，每个字符开始先发 1 个码元宽度的起始位（低电平），接着是 7 个码元宽度的 ASCII 代码和 1 个码元宽度的校验位，最后是 1～2 个码元宽度的停止位（高电平）。接收端检测由 1～2 个码元宽度的高电平跳变到 1 个码元宽度的低电平的这一特殊标志，来确定一个字符的起始位置，从而实现群同步。

（2）特定码组同步法

这是一种利用特定码组（特定的若干比特组合）来实现异步传输的方法。被传输的单位是若干比特组成的数据块（包括控制信息和数据信息），以一个特定码组作为数据块的开始标志和结束标志。因此，群是由作为群内容的数据块和首尾特定码组构成的。在传输过程中，接收端通过识别该特定码组来实现群同步。在 HDLC 规程中，这个特定码组为 01111110，在群内容中应禁止出现这个特定的比特序列。采用“0 比特插入、删除”技术，可使其具有透明传输的特性。

（3）特定字符同步法

这是一种利用特定字符作为同步标志来实现异步传输的方法。被传输的单位是由包括控制字符和数据字符在内的字符序列构成的数据块，以两个或两个以上的特定字符作为数据块的开始标志。由于这种特定字符是以实现同步为目的的，故通常称为同步字符。ASCII 编码表中的传输控制字符 SYN (0010110)就是专门为同步而设置的。它不但在信息传输期间，可用于维持字符同步，而且在空闲状态下，也可作为“时间填充”之用。至于数据块内的信息格式，则与传输控制规程的具体规定有关。

4．网同步

当进行点对点无线数字通信时，有了载波同步、位同步和群同步之后，就能实现可靠的通信了。但是现代通信已由简单的点对点通信发展到多点间的通信，即构成了通信网。因此要保证通信网内各点之间的可靠通信，就必须在网内建立一个统一的时间标准，这就是网同步。

网同步的任务就是使得整个通信网各复接点的时钟频率和相位相互协调一致。实现网同步的方法主要有两类：一类是建立全网同步系统，使通信网内各站的时钟彼此同步，即各站时钟的频率和相位都保持一致。建立全网同步的主要方法有主从同步法和相互同步法。另一类是建立准同步系统，又称独立时钟法或异步复接。此时各站均单独设置高稳定性的时钟，且允许各支路的速率偏差在一定的许可范围内，这样，在复接时各支路输入速率调整到本站的速率上，再传送出去。

常用的网同步方法有：

（1）主从同步法

主从同步法是在整个网内设立一个主站，它备有一个高稳定的主时钟源，主时钟源产生的时钟信号一般按照树状结构逐级送往各从站，使得各从站的时钟直接或间接地受到主时钟的控制。各从站的时钟频率又通过各自的锁相环与主时钟源频率保持一致。各从站还设置时延调整电路去补偿因站间距离不同所带来的不同时延。这样就保证了整个网内各站时钟的同步。主从同步法对传输线路和主时钟源的可靠性有很高的要求。因为主时钟源发生故障，使全网各站都因失去同步而无法工作；而某一中间站发生故障，不但本站不能工作，其后的各从站也因失去同步而不能工作。然而，由于这种方法具有时钟稳定度高、设备简单的优点，且方法又简单易行，所以在小型通信网中得到了广泛的应用。

（2）相互同步法

为了克服主从同步法过分依赖主时钟源的缺点，提出了相互同步法。它让网内各站都设有自己的时钟，并实现网络高度互连，使各站的频率被锁定在网内各站固有频率的平均值上，从而实现全网同步。通常称此平均值为网频率。由于相互同步是一种相互控制的过

程，当网内某一站出现故障时，网频率将平滑地过渡到一个新值，使得其余站仍能正常工作，从而提高了通信网工作的可靠性。不过这一优点是以增加每一站设备的复杂性而换得的。另外，各站时钟频率的变化都会引起网频率的变化，出现暂时的不稳定，也容易引起复接误码。

（3）码速调整法

码速调整法有正码速调整、负码速调整和正/负码速调整三种。以正码速调整法（又称正脉冲塞入法）为例，对合路器前和分路器后的码速调整设备均称为复接设备，每个支路都具有收、发两个部分，并要用两路独立信道，一路传送数据信息，另一路传送指示填充脉冲位置的标志信息。在正码速调整时，合路器提供的取样时钟频率应高于各输入支路数据流的速率。

（4）水库法

水库法是依靠在通信网的各交换站设置极高稳定度的时钟源和容量足够大的缓冲寄存器，使得在很长时间间隔内不会发生“取空”或“溢出”现象，因而无需进行码速调整。但需要定期检查缓存的状态。这如同水库一样，故称为水库法。

1.5.4 差错控制技术

无线数字通信系统的基本任务是在无线信道上确保高效而无差错地传输和处理数据信息。然而无线数字通信系统的各个组成部分都存在着产生差错的可能性，所幸的是设备部分可以达到很高的可靠性和稳定度，因而一般认为数据通信中的差错主要来自于传输媒体，即数据传输信道。无线数字信号经过远距离的无线信道传输，往往会受到各种外来干扰（如宇宙噪声、工业干扰等）的影响，这种干扰将使接收到的无线数字信号出现差错。另外，传输信道本身的传输特性不理想，也会使被传输的数据信号产生失真和时延，这也是导致接收信号产生差错的原因。由上述两种原因引起的数据信号序列错误，可以归纳为两种类型：一是随机性错误，其特点是数据信号序列中前后出错位之间没有一定的关系；二是突发性错误，它反映了前后出错位之间的相关性。由于实际信道的复杂性，上述两种类型的错误往往是同时存在的。

当实际信道的差错率不能达到用户的要求时，可以采用以下两种方法予以改进。一种方法是改善信道的电性能，包括选择合适的传输信道、改善其传输特性、选用性能优良的设备等，这种方法因受到技术和经济因素的制约，不可能得到理想的结果；另一种方法是在无线数字通信链路两端采用差错控制技术。这两种方法在实际应用中都得到了足够的重视和关注。

所谓差错控制，是指对传输的数据信号进行检测错误和纠正错误，以及发现错误而不能及时纠正错误，但能加以适当处置的某些方法。香农于 1948 年发表的题为“通信的数学理论”的论文中，曾证明：“在存在噪声的信道上，可以定义一个被称为最大信息速度的通信容量，如果用低于这个通信容量的速度发送数据，则存在着某种编码方法，采用这种编码方法可以使数据的误码率变得足够小”。这个结论既说明了对抗干扰编码理论和技术进行研究的重要性，同时也指出了对传输的数据进行某种抗干扰编码将是检测和纠正错误的有效手段。

为了检测和纠正错误，目前已研究出不少方法，但其基本方法只有三种，即时间冗余

法、设备冗余法和数据冗余法。它们的目的都是为了提高传输的可靠性，只是采用的措施不同而已，其中，时间冗余法是靠占用同一设备（包括传输媒体）的冗余来换取传输可靠性的提高。设备冗余法是通过使用较多的信道，也就是依靠设备（包括传输媒体）的冗余来换取传输可靠性的提高。数据冗余法则是前两种方法的综合，通过对数据块进行某种抗干扰编码来提高传输可靠性。可见，抗干扰编码是差错控制所关心的主要问题之一。

实现差错控制的基本思想是通过对信号码元序列作某种变换，使得原来彼此独立、无相关性的信号码元之间产生某种规律性或相关性，从而在接收端可根据这种规律性来检测甚至纠正传输序列中可能出现的错误。利用不同的变换方法可构成不同的抗干扰编码和不同的差错控制方式。抗干扰编码有检错码和纠错码之分，而目前数据通信中基本的差错控制方式有以下 4 种：

（1）检错反馈重发或自动检错重发（ARQ）

采用这种方式时，发送端按编码规则对拟发送的信号码元附加冗余码元之后再发送出去，接收端对收到的信号序列进行差错检测，并通过反馈信道把检测结果回送到发送端。如接收端认为有错，则要求发送端重发原来的数据，直至接收端正确接收为止；否则，将继续发送新的数据。ARQ 方式在物理实现上必须提供一条反馈信道及相应的缓冲器和控制机构，需要重发的次数与信道的状况有关。但这种方式的检错效果较好，所附加的冗余码元约占总发送码元的 5%～20%，通常这种方式的传输效率比前向纠错要高，因此适用于对数据通信实时性无特殊要求的场合。

（2）前向纠错（FEC）

发送端按照一定的编码规则对拟发送的信号码元附加冗余码元，构成纠错码。接收端将附加冗余码元按一定的译码规则进行变换，用来检测所收到的信号中有无错码。如有错，能自动地确定错码位置并加以纠正。FEC 方式的物理实现简单，无需反馈信道，适用于实时通信系统。但是 FEC 差错控制码与信道的差错统计特性有关，因此对信道的差错统计特性必须有充分的了解。另外，冗余码元要占总发送码元的 20%～50%，从而降低了传输效率。

（3）混合纠错（HEC）

此方式是前两种方式的结合。发送端发送具有检错和纠错能力的码元，接收端对所接收的码组中的差错个数在纠错能力以内者，能自动进行纠错，否则接收端通过反馈重发的方法来纠正错误。这种方式虽综合了 ARQ 和 FEC 的优点，但未能克服各自的缺点，因而限制了它的实际应用。

（4）不用编码的差错控制

不用编码的差错控制是指无需对被传输的信号码元进行差错编码，而在传输方法中附加冗余措施来减少传输中的差错。

1．采用检错码的差错控制

下面介绍几种实用的检错码及其检错方法。这些检错码的生成方法简单，便于实现，检错效果也好，因而得到了广泛的应用。

（1）奇偶校验码

奇偶校验码是一种最简单的检错码。在信道质量较好或传输速率较低的无线数字通信中得到了普遍使用。其编码规则是：将所要传送的数据信息分组，再在一组内各信息码元后面附加一个校验码元，使得该组码元中“1”的个数为奇数或偶数。按照此规则编成的

校验码分别称为奇校验码或偶校验码。关于奇、偶校验的选择，ISO 规定：在同步传输系统中，采用奇校验；而在异步传输系统中，采用偶校验。在实际应用中，奇、偶校验分为垂直（纵向）奇偶校验、水平（横向）奇偶校验和垂直水平奇偶校验等三种。垂直奇偶校验又称字符奇偶校验，它是指在 n–1 位表示字符信息码元后面再附加一个第 n 位的校验码元。水平奇偶校验是将传输的字符分为若干个信息码组，对同一码组内的各字符的同一位进行奇偶校验，从而形成一个校验字符。垂直水平奇偶校验是将传输的字符分成若干信息码组，并对同一码组内的各字符同时进行垂直和水平奇偶校验。

（2）循环冗余校验码

循环冗余校验码简称循环码或 CRC (Cycle Redundancy Code)，是一种高性能的检错、纠错码。由于它的检错能力强、实现简单容易，因而在数据通信中得到了非常广泛的应用。循环码在实际应用中常用做检错码。循环码的特点是有严密的数学结构，对其进行分析要用到近代代数理论。循环码的校验能力与生成多项式有关。若能针对传输信息的差错模式设计生成多项式，就会得到较强的检错能力。目前人们已经设计了许多生成多项式，下面三个多项式已成为国际标准：

CRC-12：　　$g(X)=X_{12}+X_{11}+X_3+X_2+X+1$

CRC-16：　　$g(X)=X_{16}+X_{15}+X_2+1$

CRC-CCITT：　　$g(X)=X_{16}+X_{12}+X_5+1$

其中，CRC-12 用于 6 位字符同步系统，能检测出长度在 12 位以内的突发差错。CRC-16 和 CRC-CCITT 用于 8 位字符同步系统，它们能够检测出全部的 1 位、2 位和奇数位的差错，所有长度不大于 16 位的突发错，以及 99.997％的 17 位突发错和 99.998％的 18 位或更多位的突发错。

2．采用纠错码的差错控制

纠错码是一种既有检错功能又有纠错功能的抗干扰编码。这里介绍一种比较简单的纠错码——汉明码，来说明纠错的基本原理。汉明码是 1949 年由美国贝尔实验室汉明（Hamming）提出来的，是纠正单个随机错误的线性码。目前汉明码及其变型已被广泛应用于数据通信和数字存储系统中作为差错控制之用。汉明码的码型结构与循环码相同，也是码组结构，由信息码元和校验码元组成。发送端根据编码规则产生校验码元，接收端则按照译码规则找出差错的具体位置后自动进行纠正。

汉明码具有下列参数：

码长　　$n=2^r-1$

信息码元数　　$k=2^r-1-r$

检验码元数　　$r=n-k$

最小码距　　$d_{\min}=3$

其中，码距是指两个等长码组之间对应位取值不同的个数；r 是不小于 3 的正整数。

上述参数表明，如要指明单个错误的位置 ，码长与检验码元之间应满足下列关系式：

$$2^r-1\geqslant n \quad 或 \quad 2^r\geqslant n+1$$

上式称为汉明不等式。当不等式取等号时，表示码长一定，用来纠正一位错码的汉明码所用的校验码元个数最少。这说明汉明码与相同码长的用来纠一位错码的其他线性分组码相比，它的编码效率最高。汉明码的具体编码和译码方法参加第 6 章。

3．采用冗余技术的差错控制

在传输方法中采用冗余技术，也可以降低接收数据的传输差错。

（1）回程校验

回程校验又称“回声法”。它是在接收端收到数据信息后，同时将该组数据信息经反馈信道送回发送端。发送端将送回来的数据信息与原发送数据信息进行对照比较，如果完全一样，则认为传输接收无差错，可继续发送新的数据；如果比较发现两者不一致，则判断为传输接收有差错，发送端应控制重发。对照比较和控制重发可以采用人工或自动方式。根据回程校验的基本原理，这种方法存在某些校验不稳定因素。例如，接收端收到的数据信息本来没有错误，但在回送过程中出现了差错，使得发送端作出接收有错的判断，继而进行重发，这样在接收端造成数据信息的重复。还可能出现另一种情况，假设接收数据中某个码元由“1”错成“0”，而在回送过程中，恰好又是该码元由“0”错成“1”，此时发送端并不能作出接收出错的判断，但实际上接收数据中确实存在着差错。回程校验具有设备简单、实现容易的优点，但信道利用率下降 50％。根据应用环境和条件，如在通信速率较低，传输信道较好，已有反馈信道，要求控制简单，而信道利用率不是主要问题的场合，这种方法是可取的。

（2）重发多判

把同一数据信息在一个信道上发送多次，或者在不同信道上同时发送，接收端则根据所收到的数据信息的一致性来检测差错。通常接收端按表决方式来检验收到的数据信息，取多数作为正确者，如三中取二。重发多判是利用设备冗余度，并基于多数同时出错的概率较小这样一种差错控制方法。重发多判与回程校验相类似，也是以降低信道利用率来换取传输可靠性的提高。采用这种方法接收端所使用的差错校验设备比较简单，占用较多的信道带宽或多个信道是它唯一的缺点。

1.6 dsPIC33F 系列数字信号控制器在无线通信中的实例

dsPIC33F 系列数字信号控制器在无线通信中有着广泛的应用。这里我们以一个业余无线电中使用的 NUE-PSK3.1 型数字调制解调器为例，介绍 dsPIC33F 如何实现无线数字通信的相关功能的。

NUE-PSK3.1 型数字调制解调器是一款低速率、低功耗的短波无线 PSK31 数字调制解调器。PSK31 是一种业余无线电短波数字通信方式，在附录中有简单的介绍。它采用 BPSK 和 QPSK 调制方式，在 2kHz 的音频带宽内传输速率为 31.25bit/s 的字符信息。NUE-PSK3.1 型数字调制解调器有一个简单的人机接口，字符输入是一个标准的 PS2 接口，可以连接标准的 PS2 接口键盘；输出显示采用 LCD 液晶屏，显示区分为两个部分，上部分以图形的方式显示接收信号的频谱幅度，下部分显示收发的字符信息。

将该设备的音频输入输出接口与低功率 SSB（单边带）电台的音频接口相连，可以不用计算机，直接通过键盘和 LCD 显示设备与远在异地的同伴进行信息交流，NUE-PSK3.1 型数字调制解调器的实物图如图 1−28 所示。

从功能上将，NUE-PSK3.1 型数字调制解调器有以下几个特点：

- 可独立工作的、半双工的数字调制解调器。



图 1-28　NUE-PSK3.1 型数字调制解调器的实物图

- 小巧的手持式设备，无需计算机的介入。
- 与 SSB 电台之间通过音频 I/O 接口相连。
- 设备自带频谱显示和字符显示 LCD。
- 调制方式为 BPSK 和 QPSK。
- 通过 LCD 的菜单进行参数配置。
- 具有 PS2 接口可外接标准键盘。
- 可采用内部电池供电。
- 开放源代码，源代码采用 C 语言编写。

NUE-PSK3.1 型数字调制解调器的外部连接图如图 1-29 所示。

从硬件组成上看，NUE-PSK3.1 的核心元件是一片 dsPIC33FJ128MC706 数字信号控制器。

NUE-PSK3.1 的软件主要是基于 dsPIC33 数字信号控制器编写的，包含如下几个模块：

- 键盘输入模块：接收来自键盘输入的字符。
- LCD 显示模块：显示收发字符和接收信号的频谱幅度。
- 调制模块：发送信号调制。
- 解调模块：接收信号解调。
- 参数配置模块等。



图 1-29　NUE-PSK3.1 型数字调制解调器的外部连接图

第 2 章　MPLAB C30 编译器

dsPIC 数字信号控制器（DSC）系列将 DSP 应用所需的高性能和嵌入式应用所需标准单片机功能融合在一起。这些器件得到了一套完整的软件开发工具的充分支持，包括一个优化的 C 编译器、一个汇编器、一个链接器和一个归档程序/库管理器。C 编译器主要用于与 MPLAB ASM30 汇编器和 MPLAB LINK30 链接器配合工作。

本章将介绍 MPLAB C30 与 ANSI C 在如下几个方面的差别：

- 关键字差别。
- 语句差别。
- 表达式差别。
- MPLAB C30 的 C 编译器运行时环境。

2.1　MPLAB C30 与 ANSI C 的差别

2.1.1　关键字差别

本节说明 ANSI C 和 MPLAB C30 在关键字方面的差别。新关键字是基本 GCC 实现的一部分， 本节的讨论基于标准的 GCC 文档，选择 GCC MPLAB C30 部分的特定语法和语义来讲述。

1．指定变量的属性

MPLAB C30 的关键字__attribute__用来指定变量或结构位域的特殊属性。关键字后的双括号中的内容是属性说明。目前支持的变量属性如下：address (addr)、aligned (alignment)、deprecated、far、mode (mode)、near、noload、packed、persistent、reverse (alignment)、section ("section-name")、sfr (address)、space (space)、transparent_union、unordered、unused、weak。

也可以通过在关键字前后使用_ _（双下画线）来指定属性。例如，用__aligned__代替 aligned，以避免在头文件中使用这些关键字时出现与宏同名的情况。要指定多个属性，可在双括号内用逗号将属性分隔开。例如：__attribute__ ((aligned (16), packed))。注意，一个项目中对变量属性的使用要一致。例如，如果在文件 A 中用 far 属性定义了一个变量，在文件 B 中将其声明为 extern 而不带 far，就可能导致链接错误。

（1）address (addr)

address 属性为变量指定绝对地址。这个属性不能与 section 属性同时使用。

address 属性优先。带 address 属性的变量不能存放到 auto_psv 空间，这样做会产生警告，且编译器将此变量存放到 psv 空间。

如果要将变量存放到 psv 段，地址应为程序存储器地址，例如：

```
int var __attribute__ ((address(0x800)));
```

（2）aligned (alignment)

该属性为变量指定最小的对齐方式，用字节表示。对齐方式必须是 2 的次幂。例如，下面的声明：

```
int x __attribute__ ((aligned (16))) = 0;
```

使编译器按照 16 字节分配全局变量 x。对于 dsPIC DSC 器件，这可以与访问需要对齐的操作数的 DSP 指令和寻址模式的 asm 语句配合使用。在前面的例子中，可以显式地指定希望编译器对给定变量使用的对齐方式（用字节表示）。或者可以省略对齐方式，而要求编译器为变量使用 dsPIC DSC 器件的最大有用对齐。例如，可以这样写：

```
short array[3] __attribute__ ((aligned));
```

当省略了对齐属性说明中的对齐方式时，编译器会自动地将已声明变量的对齐方式设置为目标单片机任何数据类型所使用的最大对齐方式。在 dsPIC DSC 器件中，为双字节（1 个字）。

aligned 属性只能增大对齐，但可以通过指定 packed 属性来减小对齐。

aligned 属性与 reverse 属性冲突，同时指定两者会产生错误。

（3）deprecated

deprecated 属性使得包含这一属性的声明能被编译器特别识别到。当使用 deprecated 函数或变量时，编译器会发出警告。deprecated 定义仍将被编译器执行，并被反映到目标文件中。例如，编译以下程序：

```
int __attribute__((__deprecated__)) i;
int main() {
   return i;
}
```

将产生下面的警告：

```
deprecated.c:4: warning: 'i' is deprecated (declared at deprecated.c:1)
```

在生成的目标文件中，仍以通常的方式定义了 i。

（4）far

far 属性告知编译器不必将变量分配到 near（前 8 KB）数据空间中（即变量可以分配到数据存储器中的任何地址）。

（5）mode (mode)

在变量声明中使用 mode 属性来指定与模式 mode 对应的数据类型。实际上就是允许根据变量的宽度指定整数或浮点数类型。mode 的有效值见表 2-1。这一属性对于编写可在所有 MPLAB C30 支持的目标单片机之间移植的代码很有用。

表 2-1　mode 的有效值

模　　式	宽　　度	MPLAB　C30 类型
QI	8 位	char
HI	16 位	int
SI	32 位	long
DI	64 位	long long
SF	32 位	float
DF	64 位	long double

例如，如下函数将两个 32 位有符号整数相加，并返回一个 32 位有符号整数结果：

```
typedef int __attribute__((__mode__(SI))) int32;
int32
add32(int32 a, int32 b)
  {
        return(a+b);
  }
```

可以指定 byte 或__byte__模式对应于单字节整数，word 或__word__模式对应于单字整数，pointer 或__pointer__模式用于表示指针。

（6）near

near 属性告知编译器将变量分配到 near 数据空间（数据存储器的前 8 KB）。对这种变量的访问有时比访问未分配（或不知已分配）到 near 数据空间的变量效率高。例如：

```
int num __attribute__ ((near));
```

（7）noload

noload 属性指明应该为变量分配空间，但不应为变量装入初值。这一属性对于设计在运行时将变量装入存储器（如从串行 EEPROM）的应用程序可能有用。例如：

```
int table1[50] __ attribute__ ((noload)) = { 0 };
```

（8）packed

packed 属性指定变量或结构位域采用最小的可能对齐方式：变量占①字节，位域占①位，除非用 aligned 属性指定了一个更大的值。下面的结构中位域 x 被压缩，所以它紧接在 a 之后：

```
struct foo
{
 char a;
 int x[2] __ attribute__ ((packed));
};
```

注：dsPIC 器件要求字按偶数字节对齐，因此在使用 packed 属性时要特别小心，避免运行时寻址错误。

（9）persistent

persistent 属性指定在启动时变量不应被初始化或清零。具有 persistent 属性的变量可用于存储器件复位后仍保持有效的状态信息。例如：

```
int last_mode __attribute__ ((persistent));
```

（10）reverse (alignment)

reverse 属性为变量的结束地址指定最小对齐方式。对齐以字节指定，必须是 2 的次幂。反向对齐的变量可用于递减 dsPIC DSC 汇编语言中的模缓冲区。如果应用程序需要使在 C 中定义的变量，可从汇编语言访问，这一属性可能有用。例如：

```
int buf1[128] __attribute__ ((reverse(256)));
```

reverse 属性与 aligned 和 section 属性冲突。如试图为反向对齐的变量指定一个段，系统将忽略，并发出警告。为同一个变量同时指定 reverse 和 aligned 会产生错误。带有 reverse 属性的变量不能存放到 auto_psv 空间 （参见 space 属性或-mconst-in-code 选项）；试图这样做将导致警告，且编译器会将变量存放到 psv 空间。

（11）section ("section-name")

默认情况下，编译器将其生成的目标代码存放在 .data 段和 .bss 段中。 section 属性允许指定变量 （或函数）存放到特定的段中。例如：

```
struct array {int i[32];}
struct array buf __attribute__ ((section("userdata"))) = {0};
```

section 属性与 address 属性和 reverse 属性冲突。在这两种冲突情形下，段名将被忽略，并发出警告。这一属性还可能与 space 属性冲突。

（12）sfr (address)

sfr 属性告知编译器变量是一个特殊功能寄存器 （SFR），同时使用 address 参数指定变量的运行时地址。例如：

```
extern volatile int __attribute__ ((sfr(0x200)))u1mod;
```

为避免产生错误，需要使用 extern 说明符。

注：按照约定，仅在处理器头文件中使用 sfr 属性。为将一个普通变量定义到指定的地址，要使用 address 属性，且用 near 或 far 来指定正确的寻址模式。

（13）space (space)

一般说来，编译器在一般数据空间内分配变量。可使用 space 属性来指示编译器将变量分配到特定存储空间。space 属性接受如下参数：

- data：将变量分配到一般数据空间。可使用一般 C 语句访问一般数据空间中的变量。这是默认的分配。
- xmemory：仅适用于 dsPIC30F/33F DSC。将变量分配到 X 数据空间。可使用一般 C 语句访问 X 数据空间中的变量。例如：

```
int x[32] __attribute__ ((space(xmemory)));
```

- ymemory：仅适用于 dsPIC30F/33F DSC。将变量分配到 Y 数据空间。可使用一般 C 语句访问 Y 数据空间中的变量。例如：

```
int y[32] __attribute__ ((space(ymemory)));
```

- prog：将变量分配到程序空间中为可执行代码指定的段。程序空间中的变量不能使用一般 C 语句访问，这些变量必须由编程人员显式访问，通常通过表访问行内汇编指令，或使用程序空间可视性窗口访问。
- auto_psv：将变量分配到程序空间中为自动程序空间可视性窗口访问指定的编译器管理段。auto_psv 空间中的变量可使用一般 C 语句来读（但不能写），且变量的分配空间最大为 32KB。当指定 space(auto_psv) 时，不能使用 section 属性指定段名，任何段名将被忽略并产生警告。auto_psv 空间中的变量不能存放到特定地址或反向对齐。在启动时分配到 auto_psv 段中的变量不装入数据存储器。这一属性对于减少 RAM 的使用可能有用。
- dma：仅适用于 PIC24H MCU 和 dsPIC33F DSC。将变量分配到 DMA 存储区。可以通过一般 C 语句和 DMA 外设访问 DMA 存储区中的变量。可使用__builtin_dmaoffset()来得到用于配置 DMA 外设的正确偏移量。
- psv：将变量分配到程序空间中为程序空间可视性窗口访问指定的段。链接器将定位段，因此可以通过 PSVPAG 寄存器的设置来访问整个变量。 PSV 空间中的变量不是由编译器管理的，不能使用一般 C 语句访问。这些变量必须由编程人员显式访问，通常使用表访问行内汇编指令，或使用程序空间可视性窗口访问。
- eedata：仅适用于 dsPIC30F DSC。将变量分配到 EEData 空间。EEData 空间中的变量不能使用一般 C 语句访问。这些变量必须由编程人员显式访问，通常使用表访问行内汇编指令，或使用程序空间可视性窗口访问。

（14）transparent_union

这是属于 union 型函数参数的属性，即相应的参数可以是任何联合成员的类型，但以第一个联合成员的类型传递参数。使用 transparent 联合的第一个成员的调用约定将参数传递给函数，而不是使用联合本身的调用约定。联合的所有成员必须具有相同的机器码表示，以保证参数传递正常进行。

（15）unordered

unordered 属性表明变量存放的地址可以相对于所在 C 源文件中其他变量的位置而改变。这不符合 ANSI C，但可使链接器更好地利用存储空隙。例如：

```
const int __attribute__ ((unordered)) i;
```

（16）unused

这一变量属性表明变量可能不被使用。 MPLAB C30 不会为这种变量产生未使用变量警告。

（17）weak

weak 属性声明 weak 符号。weak 符号可能被全局定义取代。如果对外部符号的引用

使用 weak，则链接时不需要该符号。例如：

```
extern int __attribute__((__weak__)) s;
int foo() {
   if (&s) return s;
   return 0; /* possibly some other value */
}
```

在上面的程序中，如果 s 没有被其他模块定义，程序仍会链接，但不会给 s 分配地址。若条件验证 s 已被定义，就返回它的值（如果它有值的话）。否则将返回“0”值。这个特征很有用，主要用于提供与任意库链接的通用代码。

weak 属性可以应用于函数和变量，例如：

```
extern int __attribute__((__weak__)) compress_data(void *buf);
int process(void *buf) {
   if (compress_data) {
      if (compress_data(buf) == -1) /* error */
   }
   /* process buf */
}
```

在上述代码中，函数 compress_data 只有在与其他模块链接时才使用。是否使用该特性是在链接时决定的，而不是在编译时决定的。

weak 属性对定义的影响更为复杂，需要多个文件加以说明，例如：

```
/* weak1.c */
int __attribute__((__weak__)) i;

void foo() {
   i = 1;
}
/* weak2.c */
int i;

extern void foo(void);

void bar() {
   i = 2;
}
main() {
   foo();
   bar();
}
```

以上程序在 weak2.c 中对 i 的定义使符号成为强定义。链接时不会出现错误，两个 i 指向同一个存储位置。为 weak1.c 中的 i 分配存储空间，但这个空间不可访问。

不能保证两个程序里的 i 具有相同的类型，如果将 weak2.c 中的 i 改为 float 型，仍

然允许链接，但是函数 foo 的操作将无法预料。foo 将向 32 位浮点值的最低有效部分写入一个值。相反，在 weak1.c 中把 i 的 weak 定义改为 float 型，将导致灾难性结果。这样会把一个 32 位的浮点值写到 16 位的整型地址中，覆盖掉紧接在 i 之后存储的任何变量。

在只存在 weak 定义的情况下，链接器才选择为定义是不可访问的。无论符号属于什么类型，操作是相同的，函数和变量具有相同的操作。

2．指定函数的属性

在 MPLAB C30 中，可以对程序中调用的函数进行某些声明，帮助编译器优化函数调用，且更准确地检查代码。关键字_attribute_允许在声明时指定特殊的属性。关键字后面的双括号中的是属性说明。目前支持函数的下列属性：address (addr)、alias ("target")、const、far、format (archetype, string-index, first-to-check)、format_arg (string-index)、interrupt[([save (list)] [, irq(irqid)] [, altirq(altir-qid)] [, preprologue(asm)])]、near、no_instrument_function、noload、noreturn、section ("section-name")、shadow、unused、weak。

也可以通过在关键字前后使用_ _（双下画线）来指定属性（例如，用__shadow __代替 shadow）。这样使得在头文件中使用它们时不必考虑会出现与宏同名的情况。要想在声明中指定多个属性，可以在双括号内使用逗号将属性分隔开，或者在一个属性声明后紧跟另一个属性声明。

（1）address (addr)

address 属性为函数指定绝对地址。这个属性不能与 section 属性同时使用，address 属性优先。例如：

```
void foo() __attribute__ ((address(0x100))) {
...
}
```

（2）alias ("target")

alias 属性为另一个符号声明一个别名，必须指定这个符号。

使用这一属性会产生对对象的外部引用，必须在链接时解析该引用。

（3）const

许多函数除了检查自身的参数外不会检查任何其他值，只会影响其返回值。可像算术运算符一样，对这种函数进行公共子表达式删除和循环优化。这些函数应该用属性 const 来声明。例如：

```
int square (int) __attribute__ ((const int));
```

也就是说，上述假设的 square 函数的实际被调用次数即使比程序指定的次数少一些也是安全的。应该注意，如果函数有指针参数，且检查指针指向的数据，那么这种函数一定不能用 const 声明。同样，调用非 const 函数的函数通常也不能声明为 const。具有 void 返回值类型的 const 函数没有什么意义。

（4）far

far 属性告知编译器不应该用更有效的调用指令形式来调用该函数。

（5）format (archetype, string-index, first-to-check)

format 属性指定一个函数具有 printf、scanf 或 strftime 类型参数，要根据格式字符串检查这些参数的类型。例如，考虑以下声明：

```
extern int
my_printf (void *my_object, const char *my_format, ...)
        __attribute__ ((format (printf, 2, 3)));
```

以上语句使编译器检查对 my_printf 调用中的参数，确定是否与 printf 类型的格式字符串参数 my_format 一致。

参数 archetype 确定如何解释格式字符串，应该为 printf、scanf 或 strftime 之一。参数 string-index 指定哪个参数是格式字符串参数（参数从左至右编号，从 1 开始），first-to-check 指定根据格式字符串检查的第一个参数的编号。对于不能检查参数的函数（如 vprintf），指定第三个参数为 0。这种情况下，编译器仅检查格式字符串的一致性。

在上面的例子中，格式字符串（my_format）是函数 my_print 的第二个参数，从第三个参数开始检查，所以 format 属性的正确参数是 2 和 3。

format 属性允许识别以格式字符串作为参数的用户自定义函数，所以 MPLAB C30 可以检查对这些函数的调用有无错误。每当要求这种警告（使用 -Wformat）时，编译器总会检查 ANSI 库函数 printf、fprintf、sprintf、scanf、fscanf、sscanf、strftime、vprintf、vfprintf 和 vsprintf 的格式，所以不必修改头文件 stdio.h。

（6）format_arg (string-index)

format_arg 属性指定一个函数具有 printf 或者 scanf 类型的参数，修改这个函数（如将它翻译为另外一种语言），并把函数的结果传递给 printf 或 scanf 类型的函数。例如，考虑以下声明：

```
extern char *
my_dgettext (char *my_domain, const char *my_format)
        __attribute__ ((format_arg (2)));
```

上述语句使编译器检查对函数 my_dgettext 的调用中的参数，该函数的结果传递给 printf、scanf 或 strftime 类型的函数，以确定是否与 printf 类型的格式字符串参数 my_format 一致。

参数 string-index 指定哪个参数是格式字符串参数（从 1 开始）。

format-arg 属性允许识别修改格式字符串的用户定义函数，所以 MPLAB C30 可以检查对 printf、scanf 或 strftime 函数的调用，这些函数的操作数是对用户定义函数的调用。

（7）interrupt [([save(list)] [, irq(irqid)][, altirq(altirqid)] [, preprologue(asm)])]

使用这个选项来指明指定的函数是中断服务程序。当指定这个属性时，编译器将生成适用于中断服务程序的函数 prologue 和 epilogue 序列。可选的参数 save 指定函数 prologue 和 epilogue 中分别保存和恢复的变量列表。可选参数 irq 和 altirq 指定要使用的中断向量表 ID。可选参数 preprologue 指定要在编译器生成的 prologue 代码前生成的汇编代码。

（8）near

near 属性告知编译器可以使用 call 指令的更有效形式调用函数。

（9）no_instrument_function

如果指定命令行选项 -finstrument-functions，那么几乎所有用户函数的入口和出口处在编译时都会被插入 profiling 函数，而函数被指定此选项时将不执行上述操作。

（10）noload

noload 属性指明应该为函数分配空间，但不应把实际代码装入存储器。如果应用程序设计为在运行时将函数装入存储器（如 EEPROM），那么这一属性就会很有用。

```
void bar() __attribute__ ((noload)) {
...
}
```

（11）noreturn

一些标准库函数是不能返回的，例如 abort 和 exit， MPLAB C30 自动清楚这种情况。有些程序自定义了不会返回的函数，可以将这些函数声明为 noreturn 来告知编译器这种情况。

```
void fatal (int i) __attribute__ ((noreturn));
void fatal (int i)
{
   /* Print error message. */
   exit (1);
}
```

noreturn 关键字告知编译器 fatal 不会返回而不必考虑如果 fatal 返回会怎样。这可以在某种程度上优化代码，而且这样有助于避免未初始化变量的假警告。

对于 noreturn 函数，非 void 的返回值类型并没有什么意义。

（12）section ("section-name")

通常，编译器将生成的代码存放在.text 段中。但有时可能需要其他的段，或者需要将某些函数存放在特殊的段中。section 属性指定将一个函数存放在特定的段中。例如下面的声明：

```
extern void foobar (void) __attribute__ ((section (".libtext")));
```

将函数 foobar 存放在 .libtext 段中。

section 属性与 address 属性有冲突。忽略段名会导致警告。

（13）shadow

shadow 属性使编译器使用影子寄存器而不是软件堆栈来保存寄存器。该属性通常与 interrupt 属性同时使用。例如：

```
void __attribute__ ((interrupt, shadow)) _T1Interrupt (void)
```

（14）unused

unused 属性表明函数可能不会被使用。MPLAB C30 不会为这种函数发出未使用函数的警告。

（15）weak

参见 1.1.1 节相关内容。

3．内联函数

通过声明一个函数为 inline，可以指示 MPLAB C30 将这个函数的代码集成到调用函数的代码中。通常这样可避免函数调用的开销，使代码执行速度更快。另外，若任何实际的参数值为常数，它们的已知值可允许在编译时进行简化，这样不用包含所有的内联函数代码。对代码量的影响是不容易预估的。使用内联函数，机器代码量视具体情况可能更大，也有可能更小。

为将函数声明为内联，在其声明中使用 inline 关键字，例如：

```
inline int
inc (int *a)
{
   (*a)++;
}
```

函数定义中的某些用法可能使函数不适合于内联替代。这些用法包括：varargs 的使用、alloca 的使用、长度可变数据的使用，以及相对 goto 和非局部 goto 的使用。如果使用了命令行选项-winline，当标识为 inline 的函数不能被替代时，会发出警告，并给出失败原因。

在 MPLAB C30 语法中，关键字 inline 不会影响函数的链接。

当一个函数同时为 inline 和 static 时，如果对该函数的所有调用都集成到调用函数中，且从不使用该函数的地址，那么该函数自身的汇编程序代码就不会被引用。这种情况下，MPLAB C30 实际上并不输出该函数的汇编代码，除非指定命令行选项-fkeep-inline-functions。有些调用由于各种原因不能被集成 （特别是在函数定义之前的调用不能被集成，定义内的递归调用也不能被集成）。如果存在非集成的调用，那么会以通常方式将函数编译成汇编代码。如果程序引用函数的地址，也必须以通常的方式编译函数，因为它不能被内联。仅在内联函数被声明为 static，且函数定义在函数使用之前的情况下，编译器才会删除内联函数。

当 inline 函数不是 static 时，编译器必须假定其他源文件可能调用这个函数。因为全局符号只能在所有程序中定义一次，不能在其他源文件中定义该函数，所以其他源文件中的调用不能被集成。因此，非 static 的内联函数总是以通常的方式编译。

如果在函数定义中同时指定 inline 和 extern，那么定义的函数就只能用来内联。不能以通常的方式编译函数，即使显式地引用其地址，因为这种地址变成了一个外部引用，如同只是声明了函数却没有定义它一样。

4．指定寄存器中的变量

MPLAB C30 允许把几个全局变量存放到指定的硬件寄存器中。

也可以指定在其中存放普通寄存器变量的寄存器。

全局寄存器变量在整个程序执行过程中保留寄存器的值。这在程序中可能很有用，如编程语言解释程序，带有几个经常被访问的全局变量。

特定寄存器中的局部寄存器变量并不保留寄存器的值。编译器的数据流分析可以确定何时指定寄存器包含有效的值，何时可将指定寄存器用于其他用途。局部寄存器变量不使用时

其中存储的值可被删除。对局部寄存器变量的引用可以被删除、移动或简化。

如果要将汇编指令的一个输出直接写到某个特定的寄存器，那么这些局部变量有时便于扩展行内汇编的使用范围。

（1）定义全局寄存器变量

在 MPLAB C30 中，可通过以下语句来定义一个全局寄存器变量：

```
register int *foo asm ("w8");
```

其中，w8 是要使用的寄存器名。选择一个可被函数调用正常保存和恢复的寄存器（W8～W13），这样库函数就不会破坏它的值。

一个函数若可能改变一个全局寄存器变量的值，它就不能安全地被保存和恢复该变量编译的函数调用，因为这可能破坏调用函数返回时期望找到的值。因此，若一个程序片段使用了全局寄存器变量，作为该程序片段入口的函数必须显式地保存和恢复属于其调用函数的值。

库函数 longjmp 将恢复每个全局寄存器变量在 setjmp 时的值。

所有全局寄存器变量的声明必须在所有函数定义之前。如果这种声明在函数定义之后，寄存器可能被声明之前的函数用于其他用途。

全局寄存器变量不能有初值，因为可执行文件不能为一个寄存器提供初值。

（2）为局部变量指定寄存器

可以通过以下语句用一个指定的寄存器定义局部寄存器变量：

```
register int *foo asm ("w8");
```

其中，w8 是使用的寄存器名。应该注意这与定义全局寄存器变量的语法相同，但是对于局部变量，这种定义应该出现在一个函数中。

定义这种寄存器不保留寄存器的值，控制确定变量的值无效时，其他用途仍可使用这种寄存器。使用这一功能，可能使编译某些函数时可用寄存器太少。

该选项并不能保证 MPLAB C30 生成的代码始终将这一变量存放在指定的寄存器中。不可以在 asm 语句中，编写对该寄存器的显式引用，并假定它总是引用这个变量。

局部寄存器变量不使用时其分配可被删除。对局部寄存器变量的引用可以被删除、移动或简化。

5．复数

MPLAB C30 支持复数数据类型。我们可以用关键字__complex__来声明整型复数和浮点型复数。例如：__complex__ float x; 定义 x 为实部和虚部都是浮点型的变量。

__complex__ short int y; 定义 y 的实部和虚部都是 short int 型的变量。

要写一个复数数据类型的常量，使用后缀“i”或“j”（两者之一，两者是等同的）。例如，2.5fi 是__complex__float 型的，3i 是__complex__int 型的。这种常量只有虚部值，但是可以通过将其与实常数相加来形成任何复数值。

要提取复数值符号 exp 的实部，可写做__real__exp。类似地，用__imag__来提取虚部。例如：

```
__complex__ float z;
```

```
float r;
float i;
r = __real__ z;
i = __imag__ z;
```

当对复数型值使用算子“~”时，执行复数的共扼。MPLAB C30 可以采用非邻近的方式分配复数自动变量，甚至可以将实部分配到寄存器中，而将虚部分配到堆栈中，反之亦然。调试信息格式无法表示这种非邻近的分配，所以 MPLAB C30 把非邻近的复数变量描述为两个独立的非复数类型变量。如果实际变量名是 foo，那么两个假设变量命名为 foo$real 和 foo$imag。

6．双字整型

MPLAB C30 支持长度为 long int 两倍的整型数据类型。对于有符号整型，使用 long long int，而对于无符号整型，则使用 unsigned long long int。可以通过在整型上添加后缀 LL 得到 long long int 类型的整型常量，在整型上添加后缀 ULL 得到 unsigned long long int 类型的整型常量。

可以在算术运算中像使用其他整型一样使用这些类型。这些数据类型的加、减和位逻辑布尔运算是开放源代码的，但是，这些数据类型的除法与移位运算不是开放源代码的。这些不开放源代码的运算要使用 MPLAB C30 自带的特殊库函数。

2.1.2 语句差别

本节讲述 ANSI C 与 MPLAB C30 所能接受的 C 语言之间的语句差别。语句差别是基本 GCC 实现的一部分， GCC 文档，本节讨论的内容是基于标准 GCC 文档，并选择了 GCC 中针对 MPLAB C30 的特定语法和语义来进行讲述。

1．将标号作为值

可以用单目运算符“&&”获得在当前函数（或包含函数）中定义的标号的地址。值的类型为 void *。这个值为常量，并可在这种类型的常量有效的任何地方使用这个值。例如：

```
void *ptr;
...
ptr = &&foo;
```

为使用这些值，需要能跳转到值。这可通过计算 goto 语句 goto *exp; 来实现。例如：

```
goto *ptr;
```

可使用 void *类型的任何表达式。

这些常量的一个用途是初始化用做跳转表的静态数组：

```
static void *array[] = { &&foo, &&bar, &&hack };
```

然后就可以通过索引来这样选择标号：

```
goto *array[i];
```

这种标号值数组的用途与 switch 语句很类似。 switch 语句更整齐，比数组更好。 标

号值的另外一个用途是可以用在线程代码的解释程序中。解释程序函数中的标号可存储在线程代码中用于快速调度。

这种机制可能被错误使用，而跳转到其他函数的代码中。编译器不能阻止这种现象的发生，因此必须小心，确保目标地址对于当前函数有效。

2．省略操作数的条件表达式

条件表达式的中间操作数可以被省略。如果第一个操作数非零，它的值就是条件表达式的值。因此，对于以下表达式：

```
x ? : y
```

如果 x 的值非零，表达式的值就是 x 的值，否则就是 y 的值。这个例子完全等价于下面的表达式：

```
x ? x : y
```

在这个简单的例子中，省略中间操作数并不是特别有用。当能存在（如果它是一个宏参数）副作用时，省略中间操作数就变得特别有用。那么重复中间操作数将产生副作用两次。省略中间操作数使用了已经计算过的值，而不会因为重新计算而产生不希望的影响。

3．case 范围

可以在单个 case 标号中指定一个连续值的范围，例如：

```
case low … high:
```

这与各个 case 标号的适当数字有相同的作用，每个数字对应从 low 到 high 中的每个整数值。

这一功能对于 ASCII 字符码范围特别有用，例如：

```
case 'A' … 'Z':
```

注意，在 … 两边要加空格，否则它和整数一起使用时可能出现解析错误。例如要这样写：

```
case 1 … 5:
```

而不要写成

```
case 1…5:
```

2.1.3　表达式差别

前面有 0b 或 0B 的一串二进制数字 （数字“0”后跟字母“b”或“B”）视为二进制整型。二进制数字由数字“0”和“1”组成。例如，十进制数字 255 可用二进制表示为 0b11111111。

像其他整型常量一样，二进制常量可以以字母“u”或“U”为后缀来指定为无符号型。二进制常量也可以以字母“l”或“L”为后缀，指定为长整型。类似地，后缀“ll”或“LL”表示双字整型的二进制常量。

2.2 MPLAB C30 C 编译器运行时环境

本节介绍 MPLAB C30 C 编译器的运行时环境。

2.2.1 地址空间

dsPIC 器件融合了传统 PICmicro 单片机的特征（外设、哈佛架构和 RISC）以及新的 DSP 功能。 dsPIC 器件具有两个独立的存储器：

- 程序存储器（见图 2–1）包含可执行代码和常量数据。
- 数据存储器（见图 2–2）包含外部变量、静态变量、系统堆栈和数据寄存器。数据存储器由 near 数据和 far 数据组成，其中，near 数据指程序存储空间的前 8KB，far 数据指数据存储空间的上面 56KB。

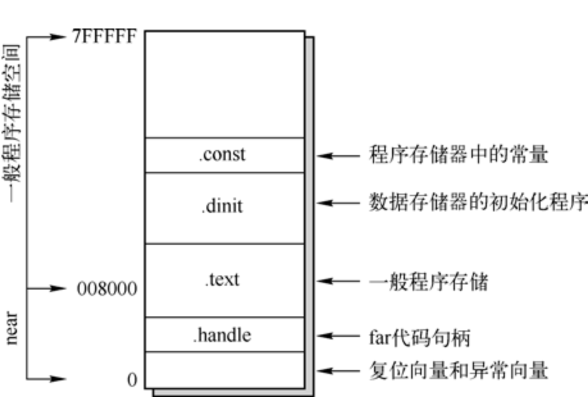


图 2–1 程序存储空间映射

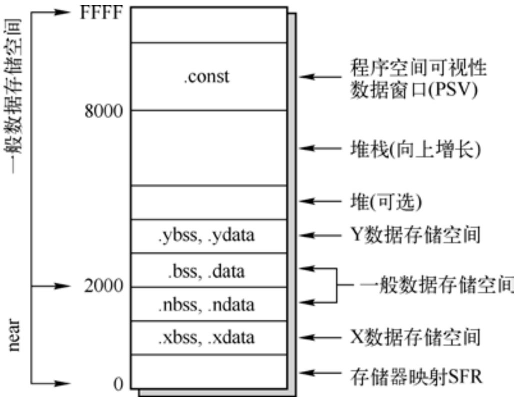


图 2–2 数据存储空间映射

尽管程序存储空间和数据存储空间是完全独立的，但编译器可通过程序空间可视性（Program Space Visibility，PSV）窗口访问程序存储空间中的常量数据。

2.2.2 代码段和数据段

段是指占用 dsPIC 器件存储器中连续地址的可定位代码或数据块。在任何给定的目标文件中，通常都有几个段。例如，一个文件中可能包含一个程序代码段、一个未初始化数据段及其他段。除非通过段属性指定，否则，MPLAB C30 编译器将代码和数据存放在默认的段中。所有由编译器生成的可执行代码都被分配到扩展名为.text 的段中，而数据则根据数据类型分配在不同的段中（见表 2–2）。

表 2–2 编译器生成的数据段

	已初始化		未初始化	
	变量	ROM 中的常量	RAM 中的常量	变量
near	.ndata	.const	.ndconst	.nbss
far	.data	.const	.const	.bss

下面列出了每个默认的段，并描述了存储到段中的信息的类型。

- text：可执行代码，分配到 .text 段中。
- data：具有 far 属性的已初始化变量，分配到 .data 段中。当选择大数据存储模型时（即使用 -mlarge-data 命令行选项时），这是已初始化变量的默认位置。
- ndata：具有 near 属性的已初始化变量，分配到 .ndata 段中。当选择小数据存储模型时（即使用默认的 -msmall-data 命令行选项时），这是已初始化变量的默认段。
- const：常量值，如字符串常量和 const 限定的变量，当使用默认的 -mconst-in-code 命令行选项时，分配到 .const 段中。这种段位于程序存储器中并通过 PSV 窗口访问。还可以通过段属性，即不需要在命令行中使用 -mconst-in-code 选项，将变量存放到.const 段中，例如：

```
int i __attribute__((space(auto_psv)));
```

- dconst：当使用 -mlarge-data 命令行选项时，不需使用 -mconst-in-code 选项，即可将常量值，如字符串常量和 const 限定的变量分配到 .dconst 段中。 MPLAB C30 启动代码将通过从 .dinit 段复制数据来初始化这种段，除非指定了链接器选项--no-data-init。.dinit 段由链接器生成，分配到程序存储器中。
- ndconst：当使用默认的 -msmall-data 命令行选项时，不需使用 -mconst-in-code 命令行选项，即可将常量值，如字符串常量和 const 限定的变量分配到 .ndconst 段中。MPLABC30 启动代码将通过从 .dinit 段中复制数据来初始化这种段。 .dinit 段由链接器生成，位于程序存储器中。
- bss：具有 far 属性的未初始化变量，分配到 .bss 段中。当选择了大数据存储模型时（即当使用 -mlarge-data 命令行选项时），这是未初始化变量的默认位置。
- nbss：具有 near 属性的未初始化变量，分配到 .nbss 段中。当选择小数据存储模型时（即使用默认的 -msmall-data 命令行选项时），这是未初始化变量的默认位置。
- pbss：持久数据。如果应用需要将数据存储在 RAM 中而不受器件复位的影响，可以使用该段。段.pbss 分配到 near 数据存储区中，不会被 libpic30.a 中的默认启动模块修改。可使用段属性将未初始化变量存放在 .pbss 段中，例如

```
int i __attribute__((persistent));
```

为了利用持久数据存储，main()函数的开头要检测所发生复位的类型。在 RCON 复位控制寄存器中的各位可用于检测复位。详细信息请参阅《dsPIC30F 系列参考手册》。

2.2.3 启动和初始化

libpic30.a 归档/库中包含两个 C 运行时启动模块。这两个启动模块的入口点都是 __reset。链接描述文件在程序存储器的地址 0 存放了一条 GOTO __reset 指令，用于在器件复位时转移控制。

默认情况下链接主启动模块并进行以下操作：

1）使用链接器或用户定义链接描述文件提供的值对堆栈指针（W15）和堆栈指针限制寄存器（SPLIM）进行初始化。

2）如果定义了.const 段，那么将通过初始化 PSVPAG 和 CORCON 寄存器将其映射到程

序空间可视性（PSV）窗口。注意，当在 MPLAB IDE 中选择了 Constants in code space 选项或在 MPLAB C30 命令行中指定了默认的-mconst-in-code 选项时，将定义一个 .const 段。

3）读取.dinit 段中的数据初始化模板，会导致所有未初始化的段被清零，同时所有已初始化段被初始化为从程序存储器中读取的值。数据初始化模板由链接器创建，并支持“代码段和数据段”中的标准段和用户定义段。

4）调用 main 函数时不带参数。

5）如果从 main 函数返回，处理器将复位。

当指定-Wl、--no-data-init 选项时，将链接备用启动模块（crt1.o)。crt1.o 执行和主启动模块相同的操作，除了第 3 步，这一步省略掉。备用启动模块比主启动模块小，所以当不需要初始化数据时，可选择该模块以节省程序存储空间。

这两个模块的源代码 （采用 dsPIC 汇编语言）存放在 c:\pic30_tools\src 目录中。如果需要，可以对启动模块进行修改。例如，如果应用需要在调用 main 函数时带参数，可通过改变条件汇编伪指令来提供这一支持。

注意：持久数据段.pbss 不会被清零或初始化。

2.2.4 存储空间

静态和外部变量通常分配到一般数据存储区中。如果选择了 constants-in-data 存储模型，const 限定的变量将被分配到一般数据存储区中；如果选择了 constants-in-code 存储模型，const 限定的变量将被分配到程序存储器中。

为配合 dsPIC DSC 的架构特点，MPLAB C30 定义了几个专用存储空间。可通过使用 space 属性，将静态和外部变量分配到以下专用存储空间中：

- data：一般数据空间，可使用普通 C 语句访问一般数据空间中的变量。这是默认的分配。
- xmemory：X 数据地址空间， 可使用普通 C 语句访问 X 数据空间中的变量。 X 数据地址空间尤其适用于针对 DSP 的函数库或汇编语言指令。
- ymemory：Y 数据地址空间，可使用普通 C 语句访问 Y 数据空间中的变量。Y 数据地址空间尤其适用于针对 DSP 的函数库或汇编语言指令。
- prog：一般程序空间，通常保留给可执行代码。不能使用普通 C 语句访问程序空间中的变量。这些变量必须由编程人员显式访问，通常通过表访问行内汇编指令，或使用程序空间可视性窗口访问。
- const：程序空间中编译器管理的区域，用于程序空间可视性（PSV）窗口访问。const 空间中的变量可通过普通 C 语句读取，总分配空间最大为 32KB。
- psv：程序空间，用于程序空间可视性（PSV）窗口访问。 psv 空间中的变量不由编译器管理，不能通过普通 C 语句访问。这些变量必须由编程人员显式访问，通常通过表访问行内汇编指令，或使用程序空间可视性窗口访问。可通过使用 PSVPAG 寄存器的设置，访问 psv 空间中的变量。
- eedata：数据 EEPROM 空间，位于程序存储器高地址的 16 位宽非易失性存储区。eedata 空间中的变量不能使用普通 C 语句访问。这些变量必须由编程人员显式访问，通常使用表访问行内汇编指令，或使用程序空间可视性窗口访问。

2.2.5 存储模型

编译器支持几种存储模型，见表 2-3。提供了命令行选项来根据所使用的特定 dsPIC 器件和存储器类型，选择最佳的存储模型。

表 2-3 存储模型命令行选项

选　　项	存储区定义	描　　述
-msmall-data	8 KB 的数据存储区 这是默认设置	允许使用类 PIC18 指令访问数据存储器
-msmall-scalar	8 KB 的数据存储区 这是默认设置	允许使用类 PIC18 指令访问数据存储器中的标量
-mlarge-data	大于 8KB 的数据存储区	使用数据引用伪指令
-msmall-code	32KW 的程序存储区 这是默认设置	函数指针不使用跳转表。函数调用使用 RCALL 指令
-mlarge-code	大于 32KW 的程序存储区	函数指针使用跳转表。函数调用使用 CALL 指令
-mconst-in-data	位于数据存储器中的常量	由启动代码从程序存储器中复制的值
-mconst-in-code	位于程序存储器中的常量 这是默认设置	通过程序空间可视性（PSV）数据窗口访问这些值

命令行选项适用于所有被编译的模块。各个变量和函数可以声明为 near 或 far，以便更好地控制代码的生成。

1．near 数据和 far 数据

如果变量分配到 near 数据段中，通常编译器能生成更好（更紧凑）的代码。如果一个应用的所有变量能存放在 8KB 的 near 数据存储区中，那么编译器在编译每个模块时就会被要求使用默认的 -msmall-data 命令行选项来将这些变量存放在 near 数据存储区中。如果标量类型（非数组或结构类型）所占用的数据总量小于 8KB，可使用默认的 -msmall-scalar 选项。这要求编译器仅将应用中的标量存放在 near 数据段中。

如果这些全局选项都不适合，那么就使用下面的可选方案：

1）可以使用-mlarge-data 或-mlarge-scalar 命令行选项编译应用的某些模块。在这种情况下，仅这些模块使用的变量被分配到 far 数据段中。如果使用这个可选方案，在使用外部定义的变量时一定要小心。如果使用这两个命令行选项之一编译的模块使用了一个外部定义的变量，则定义这个变量的模块也要使用相同的选项编译，或者在变量声明和定义时指定 far 属性。

2）如果使用了-mlarge-data 或-mlarge-scalar 命令行选项，那么可通过指定 near 属性将变量排除在 far 数据空间外（即存放在 near 数据空间）。

3）命令行选项的作用域仅限于模块内部，也可以不使用命令行选项，而通过指定 far 属性将变量存放在 far 数据段中。如果应用的 near 变量在 8KB 的 near 数据空间中存放不下，链接器将产生错误消息。

2．near 代码和 far 代码

具有 near 属性的函数（函数在彼此的 32KW 范围内）互相调用时比非 near 属性的函数效率高。如果已知应用程序中的所有函数都是 near 属性的，那么在编译每个模块时就可以使用默认的 -msmall-code 命令行选项来指示编译器采用更高效的函数调用形式。

如果这个默认的选项不适合，可以使用下面的可选方案：

1）可以使用 -msmall-code 命令行选项来编译应用程序的某些模块。在这种情况下，只有这些模块中的函数调用可以采用更高效的函数调用形式。

2）如果已使用-msmall-code 命令行选项，那么编译器可能被指示对具有 far 属性的函数使用长函数调用形式。

3）命令行选项的作用域限于模块内部，可以不使用命令行选项，而通过在函数的定义和声明中指定 near 属性，指示编译器使用更高效的函数调用形式调用这些函数。

-msmall-code 命令行选项与 -msmall-data 命令行选项的区别在于，采用前者时，为确保函数彼此“靠近”分配，编译器不需进行特别的操作；而采用后者时，编译器要将变量分配到特殊的段中。

如果函数声明为 near，而其调用函数无法采用函数调用的更高效形式调用它时，链接器将产生错误信息。

2.2.6 定位代码和数据

正如 2.2.2 节中所述，编译器将代码存放在 .text 段中，而根据所使用的存储模型和数据是否已初始化将数据存放在指定的段中。链接模块时，链接器根据各个段的属性来确定段的起始地址。

某些情况下必须将特定函数或变量存放在某个特定地址或某个地址范围内。为实现这一点，最简单的方法是使用 address 属性，如前述“关键字差别”所述。例如，将函数存放到程序存储器的地址 0x8000 中：

```
int __attribute__ ((address(0x8000))) PrintString (const char *s);
```

同样，将变量 Mabonga 存放到数据存储器的地址 0x1000 中：

```
int __attribute__ ((address(0x1000))) Mabonga = 1;
```

定位代码和数据的另一种方法是将函数或变量存放到用户定义的段中，并在自定义的链接描述文件中指定该段的起始地址。具体如下：

1）在 C 源程序中修改代码或数据的声明来指定用户定义的段。

2）将这个用户定义段加入到一个自定义的链接描述文件中来指定段的起始地址。

例如，要将函数 PrintString 存放到程序存储器的 0x8000 地址中，首先要在 C 源程序中对函数进行如下声明：

```
int __attribute__((__section__(".myTextSection")))
PrintString(const char *s);
```

段属性指定将函数存放到名为 .myTextSection 的段中，而不是默认的 .text 段中。它没有指定用户定义的段存放在哪里。这必须在一个自定义的链接描述文件中指定，以针对器件的链接描述文件为基础，则应加入如下段定义：

```
.myTextSection 0x8000 :
  {
        *(.myTextSection);
  } >program
```

这指定了输出文件应包括一个名为 .myTextSection 的段，这个段位于地址 0x8000 处，包含所有名为 .myTextSection 的输入段。由于在本例中，在该段中只有一个函数 PrintString，那么这个函数将位于程序存储器的地址 0x8000 处。

类似地，要将变量 Mabonga 存放到数据存储器的地址 0x1000 中，首先要在 C 源程序中声明该变量如下：

```
int __attribute__((__section__(".myDataSection"))) Mabonga = 1;
```

段属性指定将变量存放到名为 .myDataSection 的段中，而不是默认的 .data 段中。它没有指定用户定义的段存放在哪里。同样地，这必须在一个自定义的链接描述文件中指定，以特定器件的链接描述文件为基础，则应加入如下段定义：

```
.myDataSection 0x1000 :
  {
        *(.myDataSection);
  } >data
```

这指定了输出文件应包含一个名为 .myDataSection 的段，这个段位于地址 0x1000 处，包含所有名为.myDataSection 的输入段。在这个例子中，由于该段中仅包含一个变量 Mabonga，那么该变量将被存放到数据存储器的地址 0x1000 中。

2.2.7 软件堆栈

dsPIC 器件的寄存器 W15 专门用做软件堆栈指针。所有处理器堆栈操作，包括函数调用、中断和异常都使用软件堆栈。堆栈是向上增长的，即向高存储地址增长。

dsPIC 器件也支持堆栈溢出检测。如果堆栈指针限制寄存器（SPLIM）已被初始化，器件将对所有堆栈操作的溢出进行检测。如果发生溢出，处理器将启动一个堆栈错误异常处理，默认情况下，这将引起处理器复位。应用还可通过定义一个名为_StackError 的中断函数来安装一个堆栈错误异常处理程序。

C 运行时启动模块在启动和初始化过程中对堆栈指针（W15）和堆栈指针限制寄存器（SPLIM）进行初始化。初值通常由链接器提供，链接器尽可能在未使用的数据存储器中分配最大的堆栈。链接映射输出文件中给出堆栈的地址。通过 -stack 链接器命令行选项，应用可确保至少获得一个最小的堆栈。

另外，可以通过自定义链接描述文件中的用户定义段来分配指定大小的堆栈。在下面的示例中，0x100 字节的数据存储区保留给了堆栈，声明了用于 C 运行时启动模块的两个符号：__SP_init 和__SPLIM_init。

```
.stack :
  {
      __SP_init = .;
      . += 0x100
      __SPLIM_init = .;
    . += 8
  } >data
```

其中__SP_init 定义了堆栈指针（W15）的初值，而 __SPLIM_init 定义了堆栈指针限制寄存器（SPLIM）的初值。__SPLIM_INIT 的值应比物理堆栈限制小至少 8 个字节，以便允许堆栈错误异常处理。如果安装了堆栈错误中断处理程序，由于要考虑到中断处理程序本身的堆栈使用，__SPLIM_INIT 的值应该更小。默认的中断处理程序不需要额外使用堆栈。

2.2.8 C 编译器中堆栈的使用

C 编译器使用软件堆栈来进行如下操作：

- 分配自动变量。
- 传递函数参数。
- 在中断函数中保存处理器状态。
- 保存函数返回地址。
- 存储临时变量。
- 函数调用时保护寄存器。

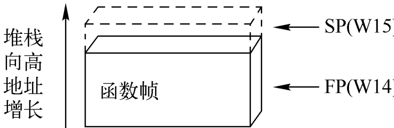


图 2−3 堆栈指针和帧指针

运行时堆栈是向上增长的，即从低地址向高地址增长。编译器使用以下两个工作寄存器来管理堆栈（见图 2−3）：

- W15：堆栈指针，指向栈顶，栈顶定义为堆栈的第一个未使用单元。
- W14：帧指针（Frame Pointer，FP），指向当前函数的帧。如果需要，每个函数都会在栈顶创建一个新的帧来分配自动变量和临时变量。可使用编译器选项-fomit-frame-pointer 来限制帧指针的使用。

C 运行时启动模块（libpic30.a 中的 crt0.o 和 crt1.o）初始化堆栈指针（W15），使其指向栈底，初始化堆栈指针限制寄存器使其指向栈顶。如果堆栈超出堆栈指针限制寄存器中的值，将转入堆栈错误陷阱。用户可以通过初始化堆栈指针限制寄存器来进一步限制堆栈的增长。

图 2−4 说明了调用一个函数的步骤。执行 CALL 或 RCALL 指令将返回地址压入软件堆栈。

现在被调用函数可以为其局部现场分配空间了，如图 2−5 所示。

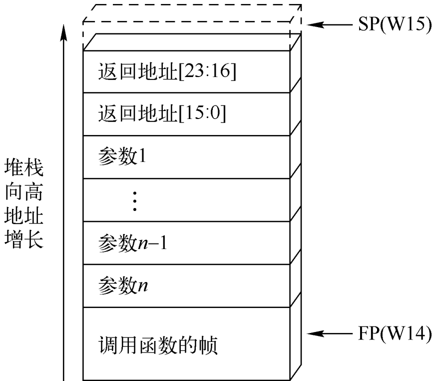


图 2−4 CALL 或 RCALL 指令

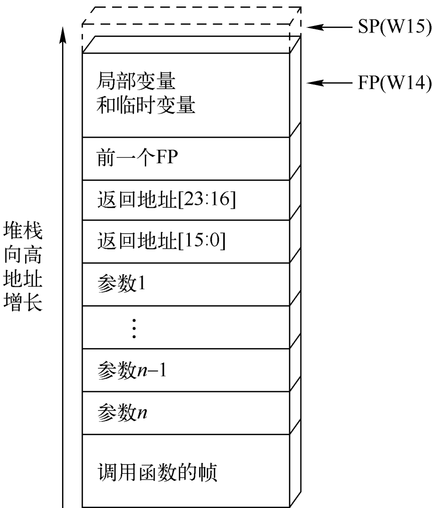


图 2−5 被调用函数的空间分配

最后，函数中用到的所有被调用函数保存寄存器被压入堆栈，如图 2–6 所示。

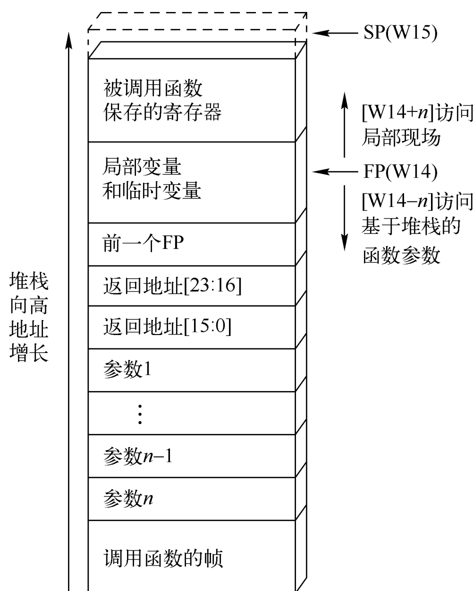


图 2–6　将被调用函数保存的寄存器弹出堆栈

2.2.9　C 编译器中堆的使用

C 运行时堆是数据存储器中的未初始化区域，用于使用标准 C 函数库中的动态存储器管理函数 calloc、malloc 和 realloc 进行动态存储器分配。如果不使用这些函数，就不需要分配堆。默认情况下不创建堆。

如果确实需要通过调用一个存储器分配函数直接使用动态存储器分配，或通过使用标准 C 函数库的输入/输出函数来间接使用动态存储器分配的话，就必须创建一个堆。通过使用链接器命令行选项 --heap 在链接器命令行中指定堆的大小即可创建一个堆。例如，使用以下命令行来分配一个 512KB 的堆：

```
pic30-gcc foo.c -Wl,--heap=512
```

则链接器会在堆栈下面分配一个堆。

如果使用标准 C 函数库的输入/输出函数，就必须分配一个堆。如果 stdout 是使用的唯一文件，那么堆的大小为 0，即使用以下命令行选项：

```
-Wl,--heap=0
```

如果打开几个文件，那么对于同时打开的每个文件，堆的大小必须包含 40 字节。如果堆存储区的空间不足，open 函数将返回错误指示。对于每个应被缓存的文件，都需要 514 字节的堆空间。如果没有足够的堆存储区空间用于缓存，文件将以非缓存模式打开。

2.2.10 函数调用约定

调用函数时有如下约定：

- 寄存器 W0～W7 由调用函数保存。为保存寄存器的值，调用函数必须将这些值压入堆栈。
- 寄存器 W8～W14 由被调用函数保存。被调用函数必须保存它会修改的任何这些寄存器。
- 寄存器 W0～W4 用于存放函数返回值。
- 如果使用-fno-short-double，则 double 等价于 long double。

参数存放到可用的第一批（个）对齐的邻近寄存器中。如果需要的话，调用函数必须保存参数。结构没有任何对齐限制；如果有足够的寄存器来保存整个结构，结构参数将占用寄存器。函数结果存储在从 W0 开始的连续寄存器中。

1．函数参数

前 8 个工作寄存器（W0～W7）用于存储函数参数。参数以自左向右的顺序分配到寄存器中，且参数被分配到对齐适当的第一个可用寄存器中。

下面的示例中，所有参数都通过寄存器传递，尽管这些参数不是以在声明中出现的顺序存放在寄存器中。这种格式允许 MPLAB C30 编译器最高效地使用可用的参数寄存器。

【例 2-1】 函数调用模型。

```
void
params0(short p0, long p1, int p2, char p3, float p4, void *p5)
{
        /*
        ** W0              p0
        ** W1              p2
        ** W3:W2           p1
        ** W4              p3
        ** W5              p5
        ** W7:W6           p4
        */
        ...
}
```

下面这个示例说明如何传递结构给函数。如果整个结构都可以存放在可用的寄存器中，那么就通过寄存器来传递结构，否则结构参数将存放在堆栈中。

【例 2-2】 函数调用模型，传递结构。

```
typedef struct bar {
  int i;
  double d;
} bar;
void
params1(int i, bar b) {
        /*
```

```
        ** W0           i
        ** W1           b.i
        ** W5:W2        b.d
        */
}
```

与长度可变参数列表中的省略号（…）对应的参数不分配到寄存器中。任何不分配到寄存器的参数都以自右向左的顺序压入堆栈。

下面的示例中，由于结构参数太大而不能存放到寄存器中。但是，这并不会禁止使用寄存器来存放下一个参数。

【例 2−3】 函数调用模型，基于堆栈的参数。

```
typedef struct bar {
   double d,e;
} bar;
void
params2(int i, bar b, int j) {
        /*
        ** W0           i
        ** stack        b
        ** W1           j
        */
}
```

对存放到堆栈的参数的访问与是否创建了帧指针有关。编译器一般情况下都创建帧指针（除非已另外指示编译器不要这样做），将通过帧指针寄存器（W14）访问基于堆栈的参数。在上面的示例中，通过 W14～22 访问 b。通过减去上一个 FP 的 2 字节，返回地址的 4 字节，和 b 的 16 字节，可计算出相对于帧指针的偏移量为-22

不使用帧指针时，汇编编程人员必须知晓自过程的入口开始使用了多少堆栈空间。如果没有额外使用堆栈空间，计算与上面类似。将通过 W15～W20 访问 b，其中，4 字节用于返回地址，16 字节用于访问 b 的首地址。

2．返回值

8 位或 16 位标量的函数返回值返回到 W0 中，32 位标量的函数返回值返回到 W1:W0 中，而 64 位标量的函数返回值返回到 W3:W2:W1:W0 中。结果通过 W0 间接返回，W0 由调用函数设置为包含结果值的地址。

3．调用函数时保存寄存器

对于一般的函数调用，编译器指定函数调用时保护寄存器 W8～W15。寄存器 W0～W7 可用做暂存寄存器。对于中断函数，编译器指定保护所有必需的寄存器，即 W0～W15 和 RCOUNT。

2.2.11 寄存器约定

特定寄存器在 C 运行时环境中起着特殊的作用。寄存器变量使用一个或多个工作寄存器，参见表 2−4。

表 2-4　寄存器约定

变　　量	工作寄存器
char，signed char，unsigned char	W0～W13 和 W14（如果没有用做帧指针的话）
short，signed short，unsigned short	W0～W13 和 W14（如果没有用做帧指针的话）
int，signed int,unsigned int	W0～W13 和 W14（如果没有用做帧指针的话）
void * (or any pointer)	W0～W13 和 W14（如果没有用做帧指针的话）
long，signed long，unsigned long	一对邻近的寄存器，第一个寄存器是 {W0, W2, W4, W6, W8, W10, W12} 之一。低编号的寄存器包含值的最低 16 位
long long, signed long long, unsigned long long	4 个邻近的寄存器，第一个寄存器是 {W0, W4, W8}之一。低编号的寄存器包含值的最低 16 位。接着的较高编号寄存器包含接着的较高位
Float	一对邻近的寄存器，第一个寄存器是 {W0, W2, W4, W6, W8, W10, W12} 之一。低编号的寄存器包含值的最低 16 位
double*	4 个邻近的寄存器，第一个寄存器是 {W0, W2, W4, W6, W8, W10, W12} 之一。低编号的寄存器包含值的最低 16 位
long double	4 个邻近的寄存器，第一个寄存器是 {W0, W4, W8}之一。低编号的寄存器包含值的最低 16 位

注意：如果使用了-fno-short-double，则 double 等价于 long double。

2.2.12　位反转寻址和模寻址

编译器并不直接支持位反转寻址和模寻址的使用。如果对一个寄存器使能了这两种寻址模式之一，那么编程人员要确保编译器不将该寄存器用做指针。当使能这两种寻址模式之一时，在产生中断时要特别小心。

可以在 C 中定义将在存储器中对齐适当的数组，用于通过汇编语言函数进行模寻址。aligned 属性可用于定义用做递增模缓冲区的数组。reverse 属性可用于定义用做递减模缓冲区的数组。

2.2.13　程序空间可视性的使用

默认情况下，编译器自动将字符串和 const 限定的未初始化变量分配到映射到程序空间可视性（PSV）窗口的.const 段。然后 PSV 管理由编译器进行管理，编译器不会移动 PSV，将可访问程序存储区的大小限制为 PSV 窗口本身的大小。或者，应用程序可控制 PSV 窗口来完成自己的功能。由应用程序直接控制 PSV 比将一个.const 段永久映射到 PSV 窗口更为灵活，但是这种情况下应用程序必须管理 PSV 控制寄存器和位。指定-mconst-in-data 选项来指示编译器不要使用 PSV 窗口。space 属性可用来定义在 PSV 窗口中使用的变量。属性 space（auto_psv）可用来指定某些变量分配到编译器管理的段.const 中。属性 space（psv）可用来将用于 PSV 访问的变量分配到不由编译器管理的段中。

第 3 章　数字滤波器的设计与实现

3.1　数字滤波的基本概念

从傅里叶分析理论知道，信号波形是由许多不同频率、不同幅度和不同初相位的正弦波叠加构成的。滤波就是提取输入信号中有用的频率成分、抑制无用的成分的信号处理过程。数字滤波是通过数值运算对输入信号序列进行滤波的数字信号处理。

3.1.1　时域离散信号

信号通常是一个自变量或几个自变量的函数。

如果信号的自变量和函数值都取连续值，则称这种信号为模拟信号或者时域连续信号，例如语言信号、电视信号等；如果自变量取离散值，而函数值取连续值，则称这种信号为时域离散信号，这种信号通常来源于对模拟信号的采样；如果信号的自变量和函数值均取离散值，则称为数字信号。数字信号也可以说成是信号幅度离散化了的时域离散信号。

时域离散信号可以用序列来表示，即离散信号可以从模拟信号采样得到，样值用 $x(n)$ 表示， $x(n)$ 为 x 的第 n 个样值。 n 不是整数时， x 未必是零，只是没有定义（是可以插值的）。 $x(n)$ 也可以本身是离散信号或由系统内部产生，在处理过程中只要知道样值的先后顺序即可，所以可以用序列来表示时域离散信号，它们的一般项为 $x(n)$ ，即

$$\begin{aligned} x &= \{x(n)\} \qquad -\infty < n < \infty \\ &= \{\cdots, x(-2), x(-1), x(0), x(1), x(2), \cdots\} \end{aligned} \tag{3-1}$$

为简便起见，常用一般项 $x(n)$ 表示序列，称为序列 $x(n)$ 。

信号随 n 的变化规律可以用公式表示，也可以用图形表示。如果 $x(n)$ 是通过观测得到的一组离散数据，则其可以用集合符号表示，例如：

$$x(n) = \{\cdots, 1.3, 2.5, 3, 1.8, 0, \cdots\}$$

3.1.2　线性时不变系统

时域离散系统的作用是将输入序列转变为输出序列，系统的功能是将输入 $x(n)$ 转变为所需输出 $y(n)$ 的运算，记为

$$y(n) = T[x(n)] \tag{3-2}$$

时域离散系统的作用如图 3−1 所示。

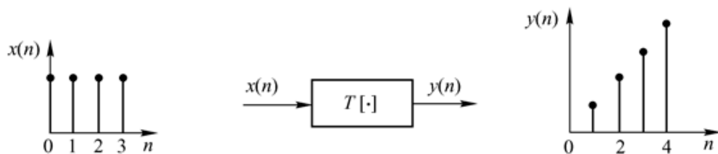


图 3-1　时域离散系统的作用示意图

在时域离散系统中，最常用最重要的是线性时不变系统（简称 LTI 离散系统），很多物理过程都可用这类系统表征，且便于分析。

1. 线性系统

满足叠加原理的系统称为线性系统。设 $x_1(n)$ 和 $x_2(n)$ 分别作为系统的输入序列，输出分别用 $y_1(n)$ 和 $y_2(n)$ 表示，即

$$y_1(n)=T\left[x_1(n)\right]$$

$$y_2(n)=T\left[x_2(n)\right]$$

那么，线性系统一定满足下面两个公式：

$$\begin{aligned}T\left[x_1(n)+x_2(n)\right]&=y_1(n)+y_2(n)\\T\left[ax_1(n)\right]&=ay_1(n)\end{aligned}\tag{3-3}$$

满足第一个公式称为线性系统的可加性，满足第二个公式称为线性系统的比例性或齐次性，式中 a 是常数，将以上两个式子结合起来，可表示为

$$y(n)=T\left[ax_1(n)+bx_2(n)\right]=ay_1(n)+by_2(n)\tag{3-4}$$

式中，a、b 均是常数。

2. 时不变系统

如果系统对输入信号的运算关系 $T[\cdot]$ 在整个运算过程中不随时间变化，或者说系统对于输入信号的响应与信号加于系统时间无关，则这种系统称为时不变系统，用公式表示为

$$\begin{aligned}y(n)&=T\left[x(n)\right]\\y(n-n_0)&=T\left[x(n-n_0)\right]\end{aligned}\tag{3-5}$$

式中，n_0 为任意整数。检查一个系统是否是时不变系统，就是检查其是否满足上式。

3. 线性时不变系统输入与输出之间的关系

设系统的输入 $x(n)=\delta(n)$，系统的输出 $y(n)$ 的初始状态为零，定义这种条件下系统输出称为系统的单位取样响应，用 $h(n)$ 表示。即单位取样响应就是系统对于 $\delta(n)$ 的零状态响应。用公式表示为

$$h(n)=T\left[\delta(n)\right]\tag{3-6}$$

$h(n)$ 和模拟系统中的 $h(t)$ 单位冲激响应类似，都代表系统的时域特征。

设系统的输入用 $x(n)$ 表示，则将其表示成单位序列移位加权和为

$$x(n)=\sum_{m=-\infty}^{\infty}x(m)\delta(n-m)\tag{3-7}$$

那么系统的输出为

$$y(n)=T\left[\sum_{m=-\infty}^{\infty}x(m)\delta(n-m)\right] \tag{3-8}$$

根据线性系统的叠加性质和时不变性质有

$$\begin{aligned}y(n)&=\sum_{m=-\infty}^{\infty}x(m)T\left[\delta(n-m)\right]\\&=\sum_{m=-\infty}^{\infty}x(m)h(n-m)\\&=x(n)*h(n)\end{aligned} \tag{3-9}$$

式中，符号“$*$”代表卷积运算，即线性时不变系统的输出等于输入序列和该系统的单位取样响应的卷积。

3.1.3 卷积

由上面的讨论可知，利用卷积可以计算线性时不变离散系统的零状态响应，卷积中主要运算是翻转、移位、相乘和相加，这类卷积称为序列的线性卷积，设两序列的长度分别是 N 和 M，线性卷积后的序列长度为 $(N+M-1)$。

线性卷积具有以下性质，分别用公式表示如下：

1. 交换律

$$x_1(n)*x_2(n)=\sum_{m=-\infty}^{\infty}x_1(m)x_2(n-m)=\sum_{m=-\infty}^{\infty}x_2(m)x_1(n-m)=x_2(n)*x_1(n) \tag{3-10}$$

上式可应用于线性系统的激励与系统的互换，如图 3−2 所示。

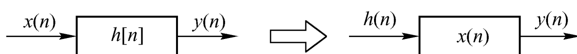


图 3−2　卷积交换律的应用

2. 分配率

$$x_1(n)*\left[x_2(n)+x_3(n)\right]=x_1(n)*x_2(n)+x_1(n)*x_3(n) \tag{3-11}$$

上式可应用于线性系统的混联组合，如图 3−3 所示。

3. 结合律

$$\begin{aligned}x_1(n)*x_2(n)*x_3(n)&=x_1(n)*\left[x_2(n)*x_3(n)\right]\\&=\left[x_1(n)*x_2(n)\right]*x_3(n)\\&=x_2(n)*\left[x_3(n)*x_1(n)\right]\end{aligned} \tag{3-12}$$

上式可应用于线性系统的级联组合，如图 3−4 所示。

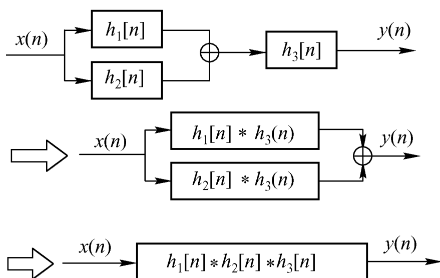


图 3-3　卷积分配律的应用

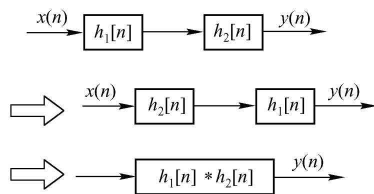


图 3-4　卷积结合律的应用

4. 任意序列与$\delta(n)$卷积

$$\delta(n)*x(n)=x(n)$$

$$\delta(n-m)*x(n)=x(n-m) \tag{3-13}$$

5. 任意序列与 $u(n)$卷积

$$u(n)*x(n)=\sum_{k=0}^{n} x(k) \tag{3-14}$$

6. 卷积的移序

$$y(n+m)=x_1(n+m)*x_2(n)=x_1(n)*x_2(n+m)$$
$$y(n+m_1+m_2)=x_1(n+m_1)*x_2(n+m_2)$$
$$y(n-m)=x_1(n-m)*x_2(n)=x_1(n)*x_2(n-m)$$
$$y(n-m_1-m_2)=x_1(n-m_1)*x_2(n-m_2)$$

3.1.4　数字滤波器的基本概念

所谓数字滤波器，是指输入、输出均为数字信号，通过一定的运算关系改变输入信号所含频率成分的相对比例或者滤除某些频率成分的器件。也就是说，它的滤波作用就是将输入信号经过某种运算或变换转变成为满足特定需要的输出信号。因此，数字滤波的概念和模拟滤波相同，只不过信号的形式和滤波的方法不同而已。

数字滤波器实际上是一个时域离散的线性时不变（LTI）系统，假设一个线性时不变系统的有理系统函数是 $H(z)$，其输入输出之间的关系可用常系数线性差分方程来描述，设某系统的系统函数为

$$H(z)=\frac{b_0+b_1Z^{-1}}{1+a_1Z^{-1}} \tag{3-15}$$

则输入 $x(n)$ 与输出 $y(n)$ 之间的关系可用常系数线性差分方程描述为

$$y(n)=-a_1y(n-1)+b_0x(n)+b_1x(n-1)$$

对于线性离散系统，可用差分方程描述输入 $x(n)$ 与输出 $y(n)$ 之间的关系，即

$$y(n)=\sum_{k=0}^{M}b_kx(n-k)-\sum_{k=1}^{N}a_ky(n-k) \tag{3-16}$$

对上式两端同时取 Z 变换，可得到系统的传输函数为

$$H(z)=\frac{\sum_{k=0}^{M}b_k z^{-k}}{1+\sum_{k=1}^{N}a_k z^{-k}} \quad (3-17)$$

数字滤波器的实现问题就是如何对上式进行运算，也就是必须把输入输出关系转换成可实现的算法。一般而言，系统的实现既可以是利用计算机软件实现的离散系统，也可以是专用硬件芯片等组成的离散系统，一般可以用英文缩写 DF 表示。前者通过算法编程对式（3−2）进行运算，而后者通过式（3−2）可确定硬件实现时需要的加法器、乘法器和延时器等基本运算单元的组合形式及个数。

与模拟滤波器比较，数字滤波器精度高，稳定可靠，灵活，便于集成化、小型化，不需要阻抗匹配，可以实现模拟滤波器无法实现的复杂滤波功能。

如果待处理的是模拟信号，可以通过 A/D 在信号形式上进行转换，再利用数字滤波器处理后经过 D/A 恢复为模拟信号。

根据单位冲激响应 $h(n)$ 的长度，可将数字滤波器可分为无限长单位脉冲响应数字滤波器（Infinite Impulse Response Digital Filter，IIR−DE）和有限长单位脉冲响应数字滤波器（Finite Impulse Response Digital Filter，FIR−DF）。这两种滤波器的设计方法和性能特点也大不相同。

数字滤波器的设计过程一般可分为以下三个阶段或过程：

- 设计系统参数的性能指标。
- 设计满足系统性能指标要求的传输函数。
- 用滤波器设计软件会硬件实现该传输函数。

3.2　IIR 滤波器

3.2.1　IIR 滤波器的基本原理和设计方法

IIR 滤波器的特点是：实现它的滤波器阶次可能较低，因此可以优先减少延迟单元和乘法器的数量，但是系统稳定性有缺陷。实现 IIR 滤波器通常采用递归算法。

设计 IIR 数字滤波器有两种相关的方法：一种是建立在已设计好的模拟低通滤波器的基础上，通过映射将模拟滤波器转变为数字滤波器。其设计步骤为：首先设计模拟滤波器，得到传输函数 $H_{\mathrm{a}}(s)$，然后将 $H_{\mathrm{a}}(s)$ 按照某种方法转换成数字滤波器的系统函数 $H(z)$。由于模拟滤波器设计方法相对成熟，不仅有完整的设计公式，还有完善的图标供查阅，故此种方法相对比较容易。另一种方法是基于直接设计技术，要用到模拟滤波器的近似数学等价条件，因此，若要以模拟低通滤波器为基础设计 IIR 数字滤波器，首先必须了解模拟低通滤波器的技术指标，同时，由于要解联立方程，设计时需要计算机进行辅助设计。

3.2.2　IIR 滤波器的 MATLAB 设计

1. 典型 IIR 滤波器的设计

MATLAB 中 IIR 数字滤波器的设计步骤如下：

1）按一定规则将给出的数字滤波器的技术指标转换成模拟低通滤波器的技术指标。

2）根据转换后的技术指标使用滤波器的阶数选择函数，确定最小的阶数 N 和固定频率 W_n 。

3）运用最小阶数 N 产生模拟滤波器原型。

4）运用固有频率 W_n 把模拟低通滤波器原型转换成模拟低通、高通、带通、带阻滤波器。

5）运用冲激响应不变法或双线性变换法把模拟滤波器转换成数字滤波器。

以上这种滤波器的设计过程称为典型的滤波器设计。

典型的滤波器的设计方法如图 3−5 所示。

下面介绍其具体实施过程。

（1）模拟原型滤波器的设计过程

模拟低通滤波器的振幅响应 $|H(\Omega)|$ 和设计规定如图 3−6 所示。通带为 $0 \sim \Omega_{\mathrm{p}}$ ，通带的幅度响应规定为 1，允许的波动为 δ_1 。阻带为 $\Omega_{\mathrm{s}} \sim \infty$ ，要求的阻带衰减为 δ_2 。 $\Omega_{\mathrm{p}} \sim \Omega_{\mathrm{s}}$ 称为过渡带，在过渡带内振幅响应不作明确规定。习惯上称频率 Ω_{p} 为通带截止频率， Ω_{s} 为阻带截止频率。为了方便起见，有时将通带截止频率归一化处理，这时，滤波器成为标准形式。

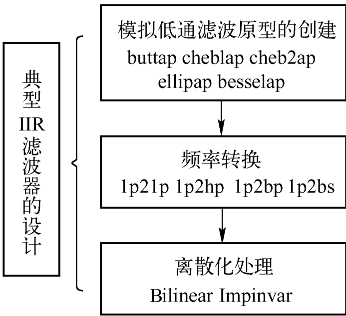


图 3−5　典型 IIR 滤波器的设计方法

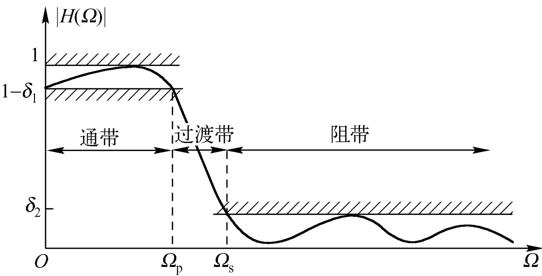


图 3−6　模拟低通滤波器的设计规定

指定低通滤波器的性能可用频率响应的幅值的平方表示。

设 $\delta_1 = 1 - \frac{1}{\sqrt{1+\varepsilon^2}}, \delta_2 = \frac{1}{A^2}$ ，则有

$$\frac{1}{\sqrt{1+\varepsilon^2}} \text{剟} |H(\Omega)|^2 \quad 1 \qquad |\Omega| \quad \Omega_{\mathrm{p}}$$

$$0 \text{剟} |H(\Omega)|^2 \quad \frac{1}{A^2} \qquad \Omega_{\mathrm{s}} \quad |\Omega|$$

其中， ε 为通带波动系数； A 为阻带衰减参数； Ω_{p} 为通带截止频率； Ω_{s} 为阻带衰减频率。

若 R_{p} 为通带允许的最大衰减， A_{s} 为阻带允许的最小衰减，则

$$R_{\mathrm{p}} = -10\lg\frac{1}{1+\varepsilon^2} \qquad \varepsilon = \sqrt{10^{R_{\mathrm{p}}/10} - 1}$$

$$A_s = -10\lg\frac{1}{A^2} \qquad A = 10^{A_s/20}$$

MATLAB 信号处理工具箱为 4 种低通模拟滤波器提供的函数分别如下。

- 巴特沃斯（Butterworth）：[Z,P,K]=buttap(n)。
- 切比雪夫Ⅰ（ChebyshevⅠ）: [Z,P,K]=cheb1ap(n,Rp)。
- 切比雪夫Ⅱ（ChebyshevⅡ）: [Z,P,K]=cheb2ap(n,Rp)。
- 椭圆法（Elliptic）：[Z,P,K]=ellipap(n,Rp,Rs)。

（2）频率转换

归一化模拟原型滤波器设计完成之后，就可以进行第二个步骤，即通过频带转换成所需要的类型（低通、高通、带通、带阻）的模拟滤波器。也就是说，模拟低通、高通、带通、带阻滤波器的主要设计方法就是先将要设计的滤波器的技术指标通过某种频率转换，转化成模拟低通滤波器的技术指标，并依据这些技术指标设计出低通滤波器的转移函数，然后再依据频率转换关系变成所要设计的滤波器的转移函数。根据模拟低通滤波器原型设计各类滤波器的频率转换式及有关设计参量表达式如表 3−1 所示。

表 3−1　频率转换法

滤波器类型	变换函数公式	相应的设计公式
低通	$z^{-1}=\dfrac{p^{-1}-\alpha}{1-\alpha p^{-1}}$	$\alpha=\dfrac{\sin\left(\dfrac{\theta_c-\omega_c}{2}\right)}{\sin\left(\dfrac{\theta_c+\omega_c}{2}\right)}$ ω_c : 截止频率
高通	$z^{-1}=\dfrac{p^{-1}+\alpha}{1+\alpha p^{-1}}$	$\alpha=\dfrac{\cos\left(\dfrac{\theta_c-\omega_c}{2}\right)}{\cos\left(\dfrac{\theta_c+\omega_c}{2}\right)}$ ω_c : 截止频率
带通	$z^{-1}=\dfrac{p^{-2}-\dfrac{2ak}{k+1}p^{-1}+\dfrac{1-k}{1+k}}{\dfrac{1-k}{1+k}p^{-2}-\dfrac{2ak}{k+1}p^{-1}+1}$	$\alpha=\dfrac{\cos\left(\dfrac{\omega_{c2}+\omega_{c1}}{2}\right)}{\cos\left(\dfrac{\omega_{c2}-\omega_{c1}}{2}\right)}$ $k=c\tan\left(\dfrac{\omega_{c2}-\omega_{c1}}{2}\right)\tan\dfrac{\theta_2}{2}$ ω_1、ω_2 : 上、下截止频率
带阻	$z^{-1}=\dfrac{p^{-2}-\dfrac{2a}{k+1}p^{-1}+\dfrac{1-k}{1+k}}{\dfrac{1-k}{1+k}p^{-2}-\dfrac{2a}{k+1}p^{-1}+1}$	$\alpha=\dfrac{\cos\left(\dfrac{\omega_{c2}-\omega_{c1}}{2}\right)}{\cos\left(\dfrac{\omega_{c2}+\omega_{c1}}{2}\right)}$ $k=\tan\left(\dfrac{\omega_{c2}-\omega_{c1}}{2}\right)\tan\dfrac{\theta_2}{2}$ ω_1、ω_2 : 上、下截止频率

【例 3−1】 用MATLAB设计一个三阶的模拟低通滤波器，通带内最大衰减为 3dB，阻带内最大衰减为 40dB，截止频率为 8π弧度，再将其转换成截止频率为 50π弧度的高通滤波器，绘出其频率响应图。

解：MATLAB 程序如下：

```
[z,p,k]=ellipap(3,3,40);
[A,B,C,D]=zp2ss(z,p,k);
[At,Bt,Ct,Dt]=lp2lp(A,B,C,D,8*pi);
[num,den]=ss2tf(At,Bt,Ct,Dt);
figure;
freqs(num,den);
[At1,Bt1,Ct1,Dt1]=lp2hp(A,B,C,D,50*pi)
[num1,den1]=ss2tf(At1,Bt1,Ct1,Dt1);
freqs(num1,den1);
```

设计的低通和高通滤波器分别如图 3-7 和图 3-8 所示。

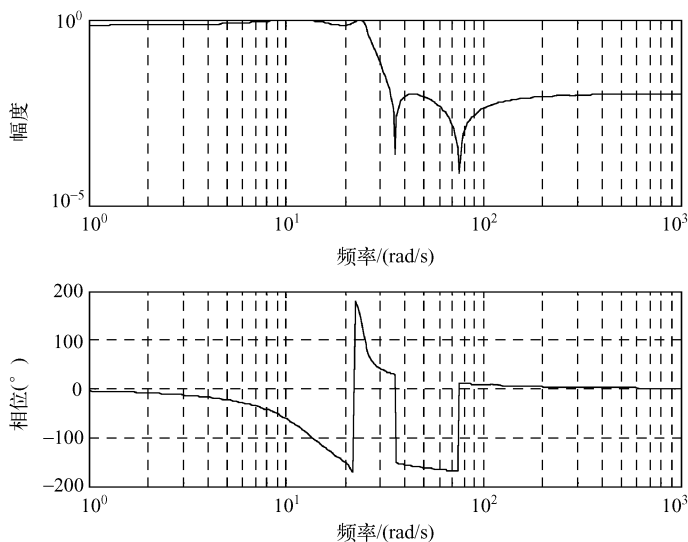


图 3-7　三阶椭圆低通滤波器的频率响应

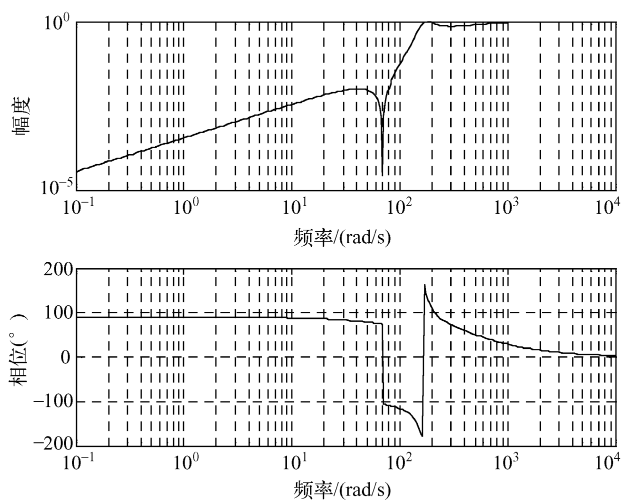


图 3-8　三阶椭圆高通滤波器的频率响应

（3）滤波器的离散化

滤波器的离散化有两种形式：冲激响应不变法和双线性变化法。

冲激响应不变法的基本原理就是从滤波器的冲激响应出发，对具有传递函数 $H(s)$ 的模型滤波器的冲激响应 $h(t)$，以周期 T 抽样所得到的离散序列 $h(nT)$ 为数字滤波器的冲激响应。

MATLAB 工具箱提供了函数 impivar( )，采用冲激响应不变法来实现模拟滤波器到数字滤波器的转换，具体的使用方法如下：

[bz,az]=impinvar(b,a,Fs)可将模拟滤波器（b,a）转换成数字滤波器（bz,az），两者的冲激响应不变，即模拟滤波器的冲激响应按 Fs 抽样后等同于数字滤波器的冲激响应。当缺少参数 Fs 时，抽样频率取默认值 1Hz。

双线性变换法：为了克服冲激响应不变法产生的频率混叠现象，需要使 s 平面与 z 平面建立一一对应的单值映射关系，即求出 $s=f(z)$，然后将它代入 $H(s)$ 就可以求出 $H(z)$，即

$$H(z)=H(s)\Big|_{s=f(z)} \tag{3-18}$$

MATLAB 工具箱提供了函数 bilinear( )，用于实现双线性变换，使用方法如下：

[zd,pd,kd]=bilinear(z,p,k,Fs)把模拟滤波器的零、极点模型转换成数字滤波器的零、极点模型，其中 Fs 是抽样频率。

[numd,dend]=bilinear(num,den,Fs)把模拟滤波器的传递函数模型转换成数字滤波器的传递函数模型，其中 Fs 是抽样函数。

[Ad,Bd,Cd,Dd]=bilinear(A,B,C,D,Fs)把模拟滤波器的状态方程模型转换成数字滤波器的状态方程模型，其中 Fs 是抽样函数。

以上三种形式，都可以增加一个参数 Fp（预畸变频率），在进行双线性变换之前，对抽样频率进行预畸变，以保证频率冲激响应在双线性变换前后具有良好的单值映射关系。

【例 3−2】 用MATLAB设计一个数字信号处理系统，其抽样频率为Fs=100Hz，在该系统设计一个巴特沃斯高通滤波器，使其通带内允许的最小衰减为 0.5dB，阻带内最小衰减为 40dB，通带上限临界频率为 30Hz，阻带下限临界频率为 40Hz。

解： MATLAB 程序实现及结果如下。

```
wp=30*2*pi;ws=40*2*pi;rp=0.5;rs=40;Fs=100;
[N,Wn]=buttord(wp,ws,rp,rs,'s');
[z,p,k]=buttap(N);
[A,B,C,D]=zp2ss(z,p,k);
[At,Bt,Ct,Dt]=lp2hp(A,B,C,D,Wn);
[num1,den1]=ss2tf(At,Bt,Ct,Dt);
[num2,den2]=bilinear(num1,den1,100);
[H,W]=freqz(num2,den2);
plot(W*Fs/(2*pi),abs(H));grid
xlabel('频率/Hz')
ylabel('幅值');
```

设计的高通滤波器如图 3−9 所示。

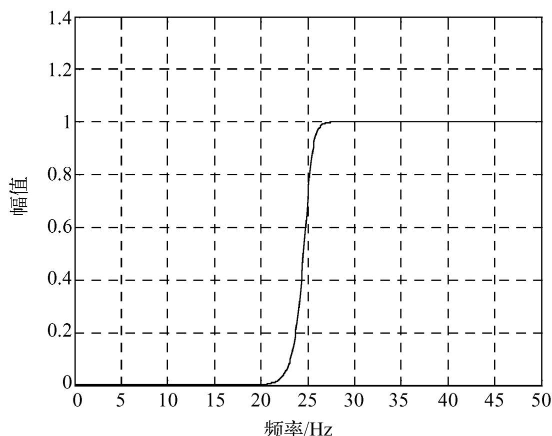


图 3-9　巴特沃斯高通滤波器的频率响应

2. 完全 IIR 滤波器的设计

在 MATLAB 信号处理工具箱中提供了几个直接设计 IIR 数字滤波器的函数，它们把以上几个步骤集成为一个整体函数，直接调用就可以设计滤波器了，这就是完全滤波器的设计。如图 3-10 所示，完全滤波器的设计为通用滤波器的设计带来了极大的方便。这些函数与数字滤波器的阶数选择函数配合使用，为特定的滤波器的设计返回所需的阶数和固有频率。

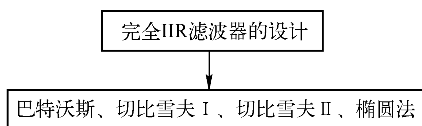


图 3-10　完全 IIR 滤波器的设计方法

这里选取巴特沃斯、切比雪夫 I、切比雪夫Ⅱ、椭圆法 4 种方法进行比较，给出用上述方法设计数字滤波器的函数如下。

- 巴特沃斯（Butterworth）滤波器：

```
[N,Wn]=buttord(Wp,Ws,Rp,Rs)
[b,a]=butter(N,Wn)
[b,a]=butter(N,Wn,'ftype')
[h,f]=freqz(b,a,n,Fs)
```

- 切比雪夫Ⅰ（ChebysheveⅠ）型滤波器：

```
[N,Wn]=cheb1ord(Wp,Ws,Rp,Rs)
[b,a]= cheb1 (N,Rp,Wn)
[b,a]= cheb1 (N,Rp,Wn, 'ftype')
[h,f]=freqz(b,a,n,Fs)
```

- 切比雪夫Ⅱ（ChebysheveⅡ）型滤波器：

```
[N,Wn]=cheb2ord(Wp,Ws,Rp,Rs)
[b,a]= cheb2 (N,Rs,Wn)
[b,a]= cheb2 (N,Rs,Wn,’ftype’)
[h,f]=freqz(b,a,n,Fs)
```

- 椭圆滤波器：

```
[N,Wn]=ellipord(Wp,Ws,Rp,Rs)
[b,a]=ellip (N,Rp,Rs,Wn)
[b,a]= ellip (N, Rp,Rs,Wn,’ftype’)
[h,f]=freqz(b,a,n,Fs)
```

其中，Wp 表示通带截止频率；Ws 表示阻带截止频率；Rp 表示通带纹波系数（通带内的最大衰减）；Rs 表示阻带纹波系数（阻带内的最小衰减）；N 表示滤波器最小阶数；Wn 表示截止频率。b、a 分别表示阶次为 N+1 的数字滤波器系统传递函数的分子和分母多项式系数向量；Fs 为采样频率；n 为在[0，Fs]频率范围内选取的频率点数；f 表示记录频率点数。n 取 2 的幂次方，可以提高运算的速度，因为 freqz 函数采用基 2 的 FFT 算法。ftype=high 时为高通滤波器，ftype=bandpass 时为带通滤波器，ftype=Stop 时为带阻滤波器。

【例 3-3】 用MATLAB设计一个 25 阶巴特沃斯滤波器，返回它的零、极点增益模型[Z,P,K]。用MATLAB实现。

分析：巴特沃斯滤波器的特点是具有通带内最平坦的幅度特性，而且随着频率的升高单调递减。因此，此滤波器又称为“最平”的幅频响应滤波器，MATLAB 信号处理工具箱为低通模拟巴特沃斯滤波器的产生提供了函数 buttap()：[Z,P,K]=buttap(n)返回一个 *n* 阶巴特沃斯滤波器的零、极点和增益。

MATLAB 实现如下：

```
[Z,P,K]=buttap(25)
```

仿真结果如下：

```
Z=
    []
P=
-0.0628+0.9980i   -0.0628-0.9980i   -0.1874+0.9823i
-0.1874-0.9823i   -0.3090+0.9511i   -0.3090-0.9511i
-0.4258+0.9048i   -0.4258-0.9048i   -0.5358+0.8443i
-0.5358-0.8443i   -0.6374+0.7705i   -0.6374-0.7705i
-0.7290+0.6845i   -0.7290-0.6845i   -0.8090+0.5878i
-0.8090-0.5878i   -0.8763+0.4818i   -0.8763-0.4818i
-0.9298+0.3681i   -0.9298-0.3681i   -0.9686+0.2487i
-0.9686-0.2487i   -0.9921+0.1253i   -0.9921-0.1253i
-1.0000
K=
  1.0000
```

3.2.3 IIR 滤波器的实例

在 NUE-PSK 型调制解调器中，广泛采用了 IIR 滤波器来实现 AGC 和符号同步中的滤波功能。

1. AGC 中的 IIR 滤波器

这两个 IIR 滤波器的 z 变换函数、幅度响应函数如下：

$$H_1(z)=\frac{z/200}{z-199/200} \tag{3-19}$$

$$H_2(z)=\frac{z/500}{z-499/500} \tag{3-20}$$

$$A_1(f)=20\lg\left(\left|\frac{1/200}{\mathrm{e}^{\mathrm{j}\frac{2\pi f}{f_s}}-199/200}\right|\right) \tag{3-21}$$

$$A_2(f)=20\lg\left(\left|\frac{1/500}{\mathrm{e}^{\mathrm{j}\frac{2\pi f}{f_s}}-499/500}\right|\right) \tag{3-22}$$

其中， $f_s=500\text{Hz}$ 。这两个滤波器的阶跃响应和幅频函数如图 4−17 所示。

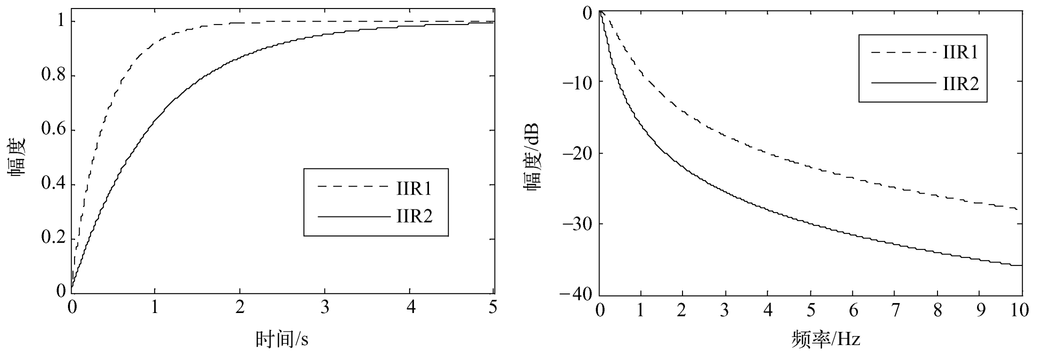


图 3−11 AGC 的 IIR 滤波器的特性

2. 符号同步中的 IIR 滤波器

在符号同步中的 IIR 滤波器的 z 变换函数、幅度响应函数如下：

$$H_1(z)=\frac{z/82}{z-81/82} \tag{3-23}$$

$$A_1(f)=20\lg\left(\left|\frac{1/82}{\mathrm{e}^{\mathrm{j}\frac{2\pi f}{f_s}}-81/82}\right|\right) \tag{3-24}$$

其中， $f_s=31.25\text{Hz}$ 。这两个滤波器的阶跃响应和幅频函数如图 3−12 所示。

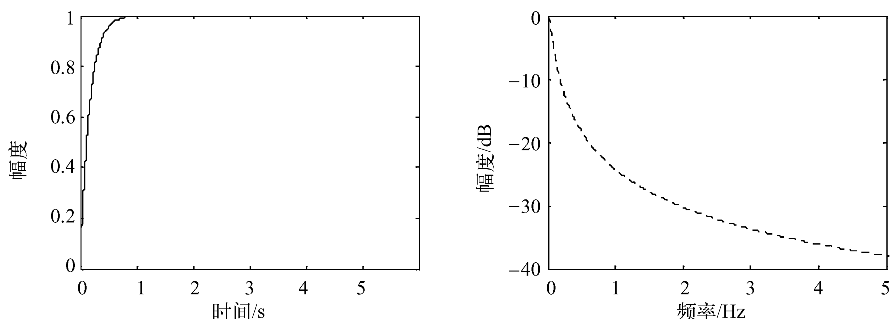


图 3–12　IIR 滤波器的特性

3.3　FIR 滤波器

IIR 数字滤波器设计利用模拟滤波器的设计成果，可以简便、有效地完成数字滤波器的设计，但是 IIR 系统幅度特性的改善一般是以相位的非线性为代价的。如果对系统有线性相位要求，IIR 系统就需要增加复杂的相位校正网络。而 FIR 系统在满足幅度特性要求下，可以保证严格的线性相位特性。

3.3.1　FIR 滤波器的基本原理和设计方法

FIR 滤波器不能直接采用由模拟滤波器的设计进行转换的方法，经常用的方法是窗函数法和频率采样法。另外还有一种比较有效的方法就是切比雪夫等波纹逼近法，需要通过计算机辅助设计完成。

数字滤波器的设计通常需要经过三个步骤来实现：

1）确定指标。根据实际应用需求确定一个滤波器的设计指标。

2）算法逼近。一旦指标确定，就应提出一个合理的滤波器模型来逼近给定的技术指标，这一步是滤波器设计面临的主要挑战。

3）系统实现。上一步的设计结果一般是得到以差分方程、系统函数或冲激响应形式描述的滤波器方程，根据这个设计方程，用滤波器设计软件来完成具体实现。

3.3.2　FIR 滤波器的 MATLAB 实现

1. 窗函数法

假设有一个截止频率为 $\omega_0\left(\mathrm{rad/s}\right)$ 的理想数字低通滤波器，其表达式如下：

$$H(\omega)=\begin{cases}1 & |\omega| \leqslant \omega_0 \\ 0 & \omega_0<|\omega| \leqslant \pi\end{cases} \tag{3–25}$$

故其冲激响应序列 $h(n)$ 为

$$h(n)=\frac{1}{2\pi}\int_{-\pi}^{\pi}H(\omega)\mathrm{e}^{\mathrm{j}\omega n}\mathrm{d}\omega=\frac{1}{2\pi}\int_{-\omega_0}^{\omega_0}\mathrm{e}^{\mathrm{j}\omega n}\mathrm{d}\omega=\frac{\omega_0}{\pi}\sin c\left(\frac{\omega_0}{\pi}n\right) \tag{3–26}$$

这个滤波器是物理不可实现的，因为其冲激响应具有无限性和非因果性。为了产生有限

区间长度的冲激响应，可以加窗函数将其截断。通过截断保留冲激响应的中心部分，就可以获得一个线性相位的 FIR 滤波器。例如，一个长度为 51、截止频率 ω_0 为 0.4rad/s 的低通滤波器，用 MATLAB 实现如下：

```
b=0.4*sinc(0.4*(−25:25));
```

此处使用的窗是一个简单的矩形窗。根据帕塞瓦尔定理，这是在最小平方积分意义上，长度为 61 的滤波器中，对理想低通滤波器的最佳近似。下面的语句可画出该滤波器的频率响应：

```
b=0.4*sinc(0.4*(−30:30));
[H,w]=freqz(b,1,512,2);
plot(w,abs(H)),grid
title('截断的 sinc 低通 FIR 滤波器')
xlabel('归一化频率（Nyquist 频率=1）')
ylabel('幅度响应')
```

其幅频响应如图 3−13 所示。

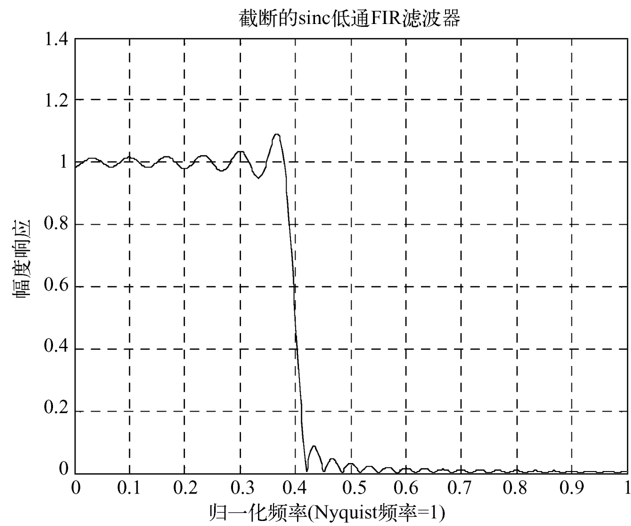


图 3−13　截断的 sinc 低通 FIR 滤波器

通过图形可以看到响应中出现的波动，尤其是在通带和阻带的边缘，这就是所谓的“吉布斯”效应，通过滤波器长度的增加是不能够消除“吉布斯”效应的。但是若采用非矩形窗就可以减小“吉布斯”效应的幅度。通过在时域内乘以窗函数，就相当于在频域产生卷积或平滑的效果，以减小“吉布斯”效应的幅度。对前面的滤波器使用长度为 61 的汉明窗：

```
b=0.4*sinc(0.4*(−30:30));
b=b.*hamming(61)';
[H,w]=freqz(b,1,512,2);
plot(w,abs(H))
grid
```

```
title('加汉明窗的 sinc 低通 FIR 滤波器')
xlabel('归一化频率（Nyquist 频率=1）')
ylabel('幅度响应')
```

其幅频响应如图 3−14 所示。

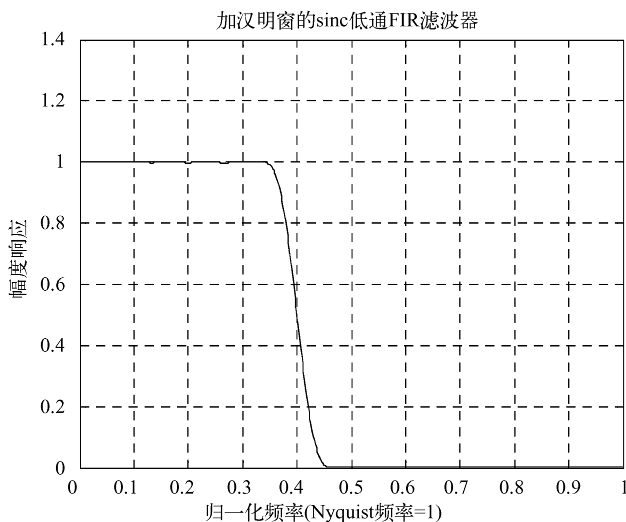


图 3−14　加汉明窗的 sinc 低通 FIR 滤波器

正如我们所见，这大大减少了“吉布斯”效应的波动，但这种改善是以牺牲过渡带宽度和最优化为代价的（加了非矩形窗后，从通带到阻带的斜坡变长了，且不再是平方积分最小了）。

2. 频率采样法

在加窗设计中，从理想滤波器的冲激响应序列着手，用截断法和加窗法就可得到所要求的滤波器频率响应。而根据频率采样法设计 FIR 滤波器时，则是从要求的滤波器频率响应出发，用内插法和离散傅里叶变换（DFT）获得滤波器的冲激响应。由此可看出，频率采样法可以拥有对任意频率响应波形滤波器的设计，具有多样性的特点，因此有很大的应用价值。

假设待设计的 FIR 滤波器的传输函数用 $H_{\mathrm{d}}(\mathrm{e}^{\mathrm{j}\omega})$ 表示，对它在 0～2π区间等间隔采样 N 点得到

$$H_{\mathrm{d}}(k)=H_{\mathrm{d}}(\mathrm{e}^{\mathrm{j}\omega})\Big|_{\omega=\frac{2\pi}{N}k},\qquad k=0,1,2,\cdots,N-1 \tag{3-27}$$

再对 N 点 $H_{\mathrm{d}}(k)$ 进行离散傅里叶逆变换（IDFT），得到 $h(n)$

$$h(n)=\frac{1}{N}\sum_{k=0}^{N-1}H_{\mathrm{d}}(k)\ \ \mathrm{e}^{\mathrm{j}\frac{2\pi}{N}nk},\qquad n=0,1,2,\cdots,N-1 \tag{3-28}$$

式中，$h(n)$ 作为所设计的滤波器的单位取样响应，其系统函数 $H(z)$ 为

$$H(z)=\frac{1-z^{-N}}{N}\sum_{k=0}^{N-1}\frac{H_{\mathrm{d}}(k)}{1-\mathrm{e}^{\mathrm{j}\frac{2\pi}{N}k}z^{-1}} \tag{3-29}$$

上式就是直接利用频率采样值 $H_d(k)$ 形成滤波器的系统函数，上面两个式子都是应用频率采样法设计的滤波器，它们分别对应不同的网络结构。

频率采样法设计滤波器最大的优点是直接从频率域进行设计，比较直观，也适合设计具有任意幅度特性的滤波器。但边界频率不易控制，如果增加采样点数 N，对确定边界频率有好处，但 N 加大会增加滤波器的成本，因此，它适合窄带滤波器的设计。

3. 等波纹线性相位滤波器的设计

理想滤波器频率响应和实际滤波器频率响应之间的差值，也就是滤波器的逼近误差，在通带或阻带区间并不是均匀分布的，一般来说靠近边缘处误差较大，离开边缘处误差较小。因此将具有误差均匀分布特性的滤波器称为等波纹滤波器。由于它在通带和阻带上的误差是均匀分布的，故需要的多项式阶次比较低。而且对于线性相位 FIR 滤波器，可以导出一组约束条件，以实现最小化最大逼近误差，也可以是最大误差最小化或 Chebyshev 最佳逼近。

3.3.3 几种重要的 MATLAB 滤波器的设计参数

几种重要的 MATLAB FIR 滤波器设计函数如表 3−2 所列。

表 3−2 常用 FIR 滤波器设计函数

方　法	MATLAB 设计函数
窗函数法	fir1, fir2, kaiserord
带过渡带的方法	firls, remez, remezord
约束最小二乘法	fircls, fircls1
任意响应滤波器设计	cremez
升余弦法	firrcos

1. 标准通带设计函数 fir1

fir1 函数实现了加窗线性相位 FIR 数字滤波器设计的经典方法，主要用于标准的通带滤波器设计，如低通、高通、带通和带阻数字滤波器设计。调用的句法有：

- b=firl(n,Wn)：返回基于汉明窗的 n 阶线性相位 FIR 低通滤波器的 n+1 个系数 b。Wn 为归一化截止频率，在[0,1]区间取值，这里 1 对应奈奎斯特（Nyquist）频率。
- b=firl(n,Wn'ftype')：指定滤波器的类型，这里的 ftype 可以是“high”（指定一个截止频率为 Wn 的高通滤波器）、“stop”（指定一个带阻滤波器）、“DC-1”（指定多带滤波器的第一频带为带通）或是“DC-0”（指定多带滤波器的第一频带为带阻）。
- b=firl(n,Wn,window)：用长度为 n+1 的列向量 window 指定窗函数，如果未给定义，则 fir1 取长度为 n+1 的汉明窗。
- b=firl(n,Wn, 'ftype',window)：是前述两种情况的综合，也就是说可以同时指定'ftype'和 window 参数。
- b=firl(…, 'normalization')：规定是否滤波器的幅度被归一化处理，字符串 normalization 可以是以下两种：

'scale' (default)：归一化滤波器，使幅度响应在通带中心频率处为 0dB。

'noscale'：不进行归一化处理。

2. 多带设计函数 fir2

fir2 函数也实现了加窗的 FIR 数字滤波器设计，但是它针对的是具有任意形状的分段线性频率响应。这个特点在 fir1 中是受限制的，因为 fir1 函数只能设计低通、高通、带通和带阻线性相位的 FIR 滤波器。fir2 函数的调用语句为：

- b=fir2(n,f,m)：返回 n 阶（n 点）FIR 滤波器的 n+1 个（行向量）系数 b。它的幅频特性与向量 f 和 m 给出的（幅频特性）相匹配。这里 f 是[0,1]区间的频点向量，1 对应于奈奎斯特（Nyquist）频率。f 的第一个点必须是 0，最后一个点必须是 1。即频点向量 f 必须是递增排列。m 是一个对应频点向量 f 的期望幅度响应向量。f 和 m 必须等长度。滤波器的形状可以用 plot(f,m)绘制。
- b=fir2(n,f,m,window)：用长度为 n+1 的列向量 window 指定窗函数。如果未指定窗，则 fir2 取长度为 n+1 的汉明窗。
- b=fir2(n,f,m,npt)：用 npt 指定点数。npt 是期望频率响应插值形成的均匀致密分布的网格。
- b=fir2(n,f,m,npt,window)：用 npt 指定点数，并且加窗。
- b=fir2(n,f,m,npt,lap)：用 lap 指定域大小。如果 f 的两个顺序值相同，则围绕这个频率，fir2 将在要求的频率响应范围内设置一个重叠的区域，以便提供一个平滑但陡峭的过渡。lap 的默认值是 25。

b=fir2(n,f,m,npt,lap,window) 用 lap 指定域大小，并且加窗。

3. 升余弦 FIR 滤波器设计函数 firrcos

firrcos 函数的主要调用句法是：

- b=firrcos(n,F0,df,fs)或 b=firrcos(n,F0,df,fs,'bandwidth')：返回具有升余弦过渡带的 n 阶 FIR 低通线性相位滤波器的系数行向量 b。
- b=firrcos(n,F0,df)：使用默认采样频率 fs=2。
- b=firrcos(n,F0,r,fs, 'rolloff ')：将括号中的第三个参量r视为衰减因子，用来代替过渡带宽df。这里r必须在[0,1]范围取值。
- b=firrcos(…, 'type')：括号中的字符串'type'的赋值可指定为'normal'或'sqrt'。前者用于设计标准的升余弦滤波器，这是系统默认的算法。后者用于设计均方根升余弦滤波器。
- b=firrcos(…'type',delay)：在[0，n+1]区间指定一个整数延迟。对偶数 n 默认延时是 n/2，对奇数 n 默认延时是（n+1）/2。
- b=firrcos(…,'type',delay,window)：对设计的滤波器进行加窗运算以减小频率响应的波纹。窗必须是长为 n+1 的列向量。

【例 3−4】 设计一个 20 阶升余弦滤波器，已知滤波器的截止频率是 0.25，过渡带宽为 0.25。

解：调用 firrcos 设计函数并绘图：

```
h=firrcos(20,0.25,0.25);
freqz(h,1)
```

设计结果如图 3−15 所示。

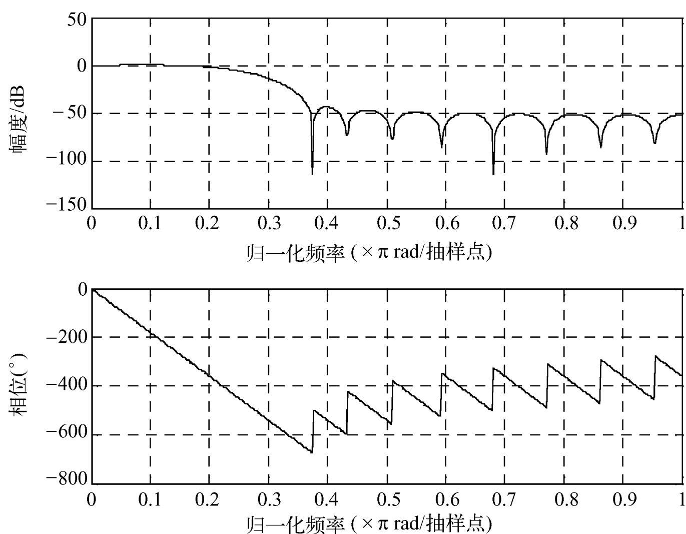


图 3-15　20 阶升余弦滤波器的频率响应

3.3.4　FIR 滤波器的 DSP 实现

在 NUE-PSK 无线通信系统中，采用 FIR 滤波器来实现匹配滤波器和自动频率跟踪（AFC）。

1. 65 抽头 FIR 匹配滤波器

在这里，FIR 滤波器有两个目的，第一个是可以在抽样点获得最大的信噪比，从而减小误码率；第二个是减小抽样点的码间串扰（Inter-Symbol Interference，ISI）。

通过精心的设计，在系统中采用以下的抽头系数：

```
x=[        32767 *   4.34544353312181E-005,
           32767 * -4.91233963259580E-004,
           32767 * -7.87728675226019E-004,
           32767 * -1.35080960413432E-003,
           32767 * -2.12882395022525E-003,
           32767 * -3.13352868120857E-003,
           32767 * -4.36690434541042E-003,
           32767 * -5.81133495736277E-003,
           32767 * -7.42514606134644E-003,
           32767 * -9.14007102453525E-003,
           32767 * -1.08603737736329E-002,
           32767 * -1.24643348445605E-002,
           32767 * -1.38080486055754E-002,
           32767 * -1.47314854128492E-002,
           32767 * -1.50673587706280E-002,
           32767 * -1.46511869827097E-002,
           32767 * -1.33336912187397E-002,
           32767 * -1.09918793481672E-002,
```

32767 * −7.54327547013411E−003,
32767 * −2.95284395921849E−003,
32767 * 2.75468424256632E−003,
32767 * 9.49346738507647E−003,
32767 * 1.71136504677581E−002,
32767 * 2.54040194354235E−002,
32767 * 3.41003627164679E−002,
32767 * 4.28966968548455E−002,
32767 * 5.14596043337083E−002,
32767 * 5.94460415073074E−002,
32767 * 6.65223329686451E−002,
32767 * 7.23830650097610E−002,
32767 * 7.67692292833128E−002,
32767 * 7.94832082001426E−002,
32767 * 8.04019187845638E−002,
32767 * 7.94832082001426E−002,
32767 * 7.67692292833128E−002,
32767 * 7.23830650097610E−002,
32767 * 6.65223329686451E−002,
32767 * 5.94460415073074E−002,
32767 * 5.14596043337083E−002,
32767 * 4.28966968548455E−002,
32767 * 3.41003627164679E−002,
32767 * 2.54040194354235E−002,
32767 * 1.71136504677581E−002,
32767 * 9.49346738507647E−003,
32767 * 2.75468424256632E−003,
32767 * −2.95284395921849E−003,
32767 * −7.54327547013411E−003,
32767 * −1.09918793481672E−002,
32767 * −1.33336912187397E−002,
32767 * −1.46511869827097E−002,
32767 * −1.50673587706280E−002,
32767 * −1.47314854128492E−002,
32767 * −1.38080486055754E−002,
32767 * −1.24643348445605E−002,
32767 * −1.08603737736329E−002,
32767 * −9.14007102453525E−003,
32767 * −7.42514606134644E−003,
32767 * −5.81133495736277E−003,
32767 * −4.36690434541042E−003,
32767 * −3.13352868120857E−003,
32767 * −2.12882395022525E−003,
32767 * −1.35080960413432E−003,
32767 * −7.87728675226019E−004,
32767 * −4.91233963259580E−004,

32767 * 4.34544353312181E-005];

对应的幅频和相频响应如图 3-16 所示。

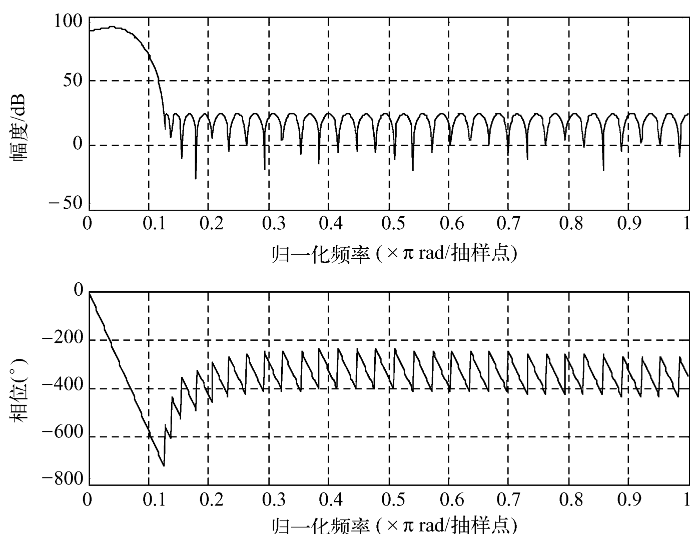


图 3-16 幅频和相频响应

抽样频率为 500Hz。

2. 65 抽头 FIR AFC 滤波器

在系统中，自动频率跟踪模块（AFC）不能利用 FIR 匹配滤波器输出的结果，因为上面的匹配滤波器的频带太窄，这样 AFC 不能锁定在 PSK31 信号的载波上。所以另外设计了一个 65 抽头的 FIR 滤波器。

它的抽头系数如下所示：

```
x=[     32767 *  -8.82290884173488E-004,
        32767 *  -1.13743887320639E-003,
        32767 *  -1.64128678489465E-003,
        32767 *  -2.05209889086253E-003,
        32767 *  -2.21756176327389E-003,
        32767 *  -1.97911605963953E-003,
        32767 *  -1.20579524827142E-003,
        32767 *   1.65272511919182E-004,
        32767 *   2.09434721909167E-003,
        32767 *   4.41226679403865E-003,
        32767 *   6.81893708889913E-003,
        32767 *   8.90459330660423E-003,
        32767 *   1.02036776754177E-002,
        32767 *   1.02663164627312E-002,
        32767 *   8.75074508195768E-003,
        32767 *   5.51366833309811E-003,
        32767 *   6.86162420348332E-004,
```

32767 * −5.28831303658911E−003,
32767 * −1.16532489911285E−002,
32767 * −1.73997493974992E−002,
32767 * −2.13766417901555E−002,
32767 * −2.24353771237607E−002,
32767 * −1.96068732388393E−002,
32767 * −1.22636654593710E−002,
32767 * −2.60786710341817E−004,
32767 * 1.59860884681409E−002,
32767 * 3.54913904978846E−002,
32767 * 5.67733050444888E−002,
32767 * 7.80023915048147E−002,
32767 * 9.72073643477517E−002,
32767 * 1.12514131590886E−001,
32767 * 1.22377761748080E−001,
32767 * 1.25803660705561E−001,
32767 * 1.22377761748080E−001,
32767 * 1.12514131590886E−001,
32767 * 9.72073643477517E−002,
32767 * 7.80023915048147E−002,
32767 * 5.67733050444888E−002,
32767 * 3.54913904978846E−002,
32767 * 1.59860884681409E−002,
32767 * −2.60786710341817E−004,
32767 * −1.22636654593710E−002,
32767 * −1.96068732388393E−002,
32767 * −2.24353771237607E−002,
32767 * −2.13766417901555E−002,
32767 * −1.73997493974992E−002,
32767 * −1.16532489911285E−002,
32767 * −5.28831303658911E−003,
32767 * 6.86162420348332E−004,
32767 * 5.51366833309811E−003,
32767 * 8.75074508195768E−003,
32767 * 1.02663164627312E−002,
32767 * 1.02036776754177E−002,
32767 * 8.90459330660423E−003,
32767 * 6.81893708889913E−003,
32767 * 4.41226679403865E−003,
32767 * 2.09434721909167E−003,
32767 * 1.65272511919182E−004,
32767 * −1.20579524827142E−003,
32767 * −1.97911605963953E−003,
32767 * −2.21756176327389E−003,
32767 * −2.05209889086253E−003,
32767 * −1.64128678489465E−003,

32767 *　 −1.13743887320639E−003,
32767 *　 −8.82290884173488E−004];

对应的幅频和相频响应如图 3−17 所示。

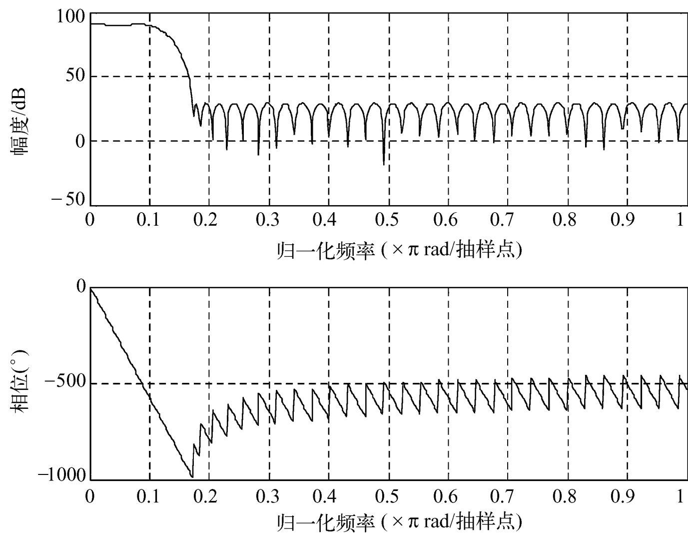


图 3−17　幅频和相频响应

抽样频率为 500Hz。

第 4 章　数字调制解调器的设计与实现

调制过程是将低频信号搬移到高频的过程。一般用被传送的低频信号来控制高频载波的某些参数（幅度、频率或相位）来实现调制。解调是个相反的过程，即把信号从高频搬移到低频的过程。一个通信系统的质量，与所采用的调制方式关系密切。调制是为了使信号特性与信道特性相匹配，因此，一定要根据系统的信道特性来选择调制方式。

4.1　无线通信中的数字调制

无线信道是典型的色散信道，有着严重的多径与衰落等不利于数据传输的特性。除此之外，通信系统中收、发两端的中频滤波器会令信道有着带限特性；发射机的功率放大器件是典型的非线性器件，因而无线信道还具有非线性的特性。

因此，一定要选择适合无线信道传输的调制方式。早期的无线通信系统采用的是模拟调制技术，而现在和未来的无线通信系统已经或将被建议采用数字调制方式。这是因为数字调制方式相对模拟调制方式有着极大的技术优势。

数字调制相对模拟调制具有更好的噪声抑制能力，具有抗干扰能力强、易于加密、可以结合纠错 / 均衡技术提高传输质量等优点。

考虑到信道的特殊性和系统实现的便利性，一般情况下，无线通信系统多采用 BPSK、QPSK 等简单的数字调制技术。数字调制解调器的实现可以采用专用的芯片，或高档的 DSP 芯片来实现。但是，考虑到经济性和灵活性，本章将介绍一种使用 Microchip 公司的 dsPIC 芯片来实现的方案。

4.1.1　无线通信系统对数字调制的要求

获得低的误码率是所有通信系统的基本要求，由于无线通信系统有着其特殊性，因此，无线通信系统对数字调制提出的要求是多方面的。需要在低信噪比、具有多径衰落的信道条件下实现低误码率，同时希望占用尽可能少的带宽，而且系统要便于实现，以节约成本，提高系统的可靠性。现有的调制方式很难同时满足上述的所有要求，实际的数字调制解调器往往是折中考虑多种因素的结果。

调制方式的性能一般用功率效率和带宽效率来衡量。功率效率是为了描述系统在较低的功率条件下，保证系统还能实现低误码率的能力。当然，为了降低系统的误码率，可以提高发射功率。但是，在很多情况下，这样做是不允许的。为此，人们用调制的功率效率 η_p 来衡量误码率与发射功率之间的折中程度，一般表示为在接收机特定的误码率条件下，每比特信号能量与噪声功率谱密度的比值（ E_b/N_0 ）。

带宽效率用来描述有限带宽内，某种调制方案传输数据的能力。一般情况下，提高数据传输速率就意味着减少每个数字符号的脉冲宽度，进而会增加信号的带宽。这样，数据传输速率与占用信号带宽之间就存在着固有的矛盾。正确地选择调制方案可以在一定程度上实现二者的较好折中。带宽效率反映了对所分配的带宽的利用程度，常被定义为每赫兹带宽上的数据吞吐量。如果用 R 表示数据传输的比特速率， B 表示已调信号所占用的带宽，则带宽效率 η_{B} （单位：bit/（s • Hz））可以表示为

$$\eta_{\mathrm{B}}=\frac{R}{B} \tag{4-1}$$

从式（4−1）可以看出，带宽效率越高，在给定的频带内传输的数据越多，但是这个带宽效率是否可以无限地接近 100%呢？

香农的信道编码理论表明，带宽效率有一个上限。在加性高斯白噪声的信道条件下，在一个任意小的错误概率条件下，带宽效率的最大值受到信道噪声的限制，信道容量的最大值表示如下：

$$\eta_{\mathrm{B\,max}}=\frac{C}{B}=\log_2\left(1+\frac{S}{N}\right) \tag{4-2}$$

其中， C 是信道容量，单位是 bit/s； B 是信道带宽，单位是 Hz； S/N 是信噪比。

在调制方式的选择上，通常需要在带宽效率和功率效率之间进行平衡。比如，对信息进行差错控制编码，一种方式是提高占用带宽，当然也就降低了系统的带宽效率，但同时给定的误比特率所需要的信号功率也降低了，这样就得到了较好的功率效率。可以这样说，用带宽效率换取了功率效率。另外一种方式，可以采用多进制的调制方式，可以降低占用的带宽，其代价是必须增加信号的功率，就相当于是用功率效率换取了带宽效率。

4.1.2 数字信号的带宽和功率谱密度

虽然数字信号的带宽没有一个适用于所有情况的定义。但是，所有的定义都是基于信号功率谱密度的某种度量。随机信号 $x(t)$ 的功率谱密度可以定义为

$$P(f)_x=\lim_{T\to\infty}\left(\frac{\overline{\left|X_{\mathrm{T}}(f)\right|^2}}{T}\right) \tag{4-3}$$

其中，上划线表示统计平均， $X_{\mathrm{T}}(f)$ 是 $x_{\mathrm{T}}(t)$ 的傅里叶变换， $x_{\mathrm{T}}(t)$ 是 $x(t)$ 的截断形式，定义如下：

$$x_{\mathrm{T}}(t)=\begin{cases}x(t)\;, & -T/2<t<T/2\\ 0\;, & \text{其他}\end{cases} \tag{4-4}$$

已调信号的功率谱密度与相应的基带复包络信号的功率谱密度有关。如果一个已调信号的 $s(t)$ 的表达式是

$$s(t)=\mathrm{Re}\left[g(t)\mathrm{e}^{\mathrm{j}2\pi f_{\mathrm{c}}t}\right] \tag{4-5}$$

其中， $g(t)$ 是基带信号的复包络，则已调信号的功率谱密度可以表示为

$$P_s(f)=\frac{1}{4}\left[P_g(f-f_c)+P_g(-f-f_c)\right] \quad (4-6)$$

这里，$P_g(f)$ 是 $g(t)$ 的功率谱密度。

通常，信号的绝对带宽定义为信号的非零值功率谱在频域上所占的频率范围。如果基带信号是矩形脉冲，其对应的功率谱密度是 $(\sin f)^2/f^2$ 类型分布，在频率轴上表现为无限延伸的特性，其带宽将是无限大的。为了能够尽可能客观地反映信号所占用的频率范围，被广泛采用的带宽度量办法是零点—零点带宽，这相当于频谱主瓣的宽度。另一种比较常用的带宽度量方法是，功率谱密度下降到峰值的一半时对应的频率间隔，被称为 3dB 带宽。

4.2 调制

调制就是用基带信号对载波波形的某些参量进行控制，使信息携带到载波的这些参量上，即所谓的载波调制。一般采用正弦波作为载波。在数字振幅调制（ASK）、数字频率调制（FSK）和数字相位调制（PSK）当中，PSK 是综合性能比较好的调制方式。而 BPSK 和 QPSK 则是数字相位调制中最简单、最基本的调制方式。下面，我们将用 dsPIC 来实现这两种比较简单而实用的调制。

4.2.1 调制器的基本原理

无论是 BPSK 还是 QPSK，其调制信号都可以表示为

$$e_{\mathrm{PSK}}(t)=\sum_n g(t-nT_s)\cos(\omega_c t+\varphi_n) \quad (4-7)$$

式中，$g(t)$为信号包络波形，理论上为矩形波，幅度为 1；T_s为码元时间宽度；ω_c为载波角频率；φ_n为第n个码元对应的相位，对于BPSK有两种取值，对于QPSK有 4 种取值。

这样的调制信号也可以表示为正交形式，即

$$\begin{aligned}e_{\mathrm{PSK}}(t)&=\left[\sum_n g(t-nT_s)\cos\varphi_n\right]\cos\omega_c t-\left[\sum_n g(t-nT_s)\sin\varphi_n\right]\sin\omega_c t\\&=\left[\sum_n a_n g(t-nT_s)\right]\cos\omega_c t-\left[\sum_n b_n g(t-nT_s)\right]\sin\omega_c t\\&=I(t)\cos\omega_c t-Q(t)\sin\omega_c t\end{aligned} \quad (4-8)$$

式中，$a_n=\cos\varphi_n$；$b_n=\sin\varphi_n$；$I(t)=\sum_n a_n g(t-nT_s)$；$Q(t)=\sum_n b_n g(t-nT_s)$。

数字相位调制信号的功率谱如图 4−1 所示，图中给出了信息速率相同时，2PSK 和 4PSK 信号的单边功率谱。可以看出，4PSK 的功率谱主瓣窄，因此频带利用率较高。

QPSK 的每个码元含有 2 比特的信息，用 a、b 代表这两个比特，则共有 4 种组合，即 00、01、10 和 11。它们和相位φ_n之间的关系通常都按格雷（Gray）码的规律变化，如表 4−1 所示。表中给出了两种编码方式，其矢量图如图 4−2 所示。

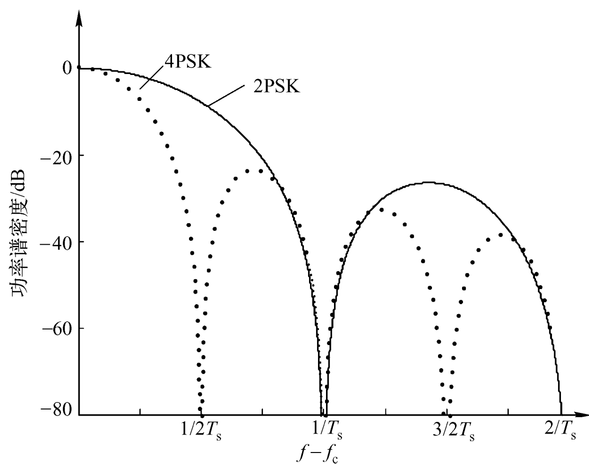


图 4−1 数字相位调制信号功率谱

表 4−1 QPSK 编码规则

a	b	φ_n
1	1	45°
0	1	135°
0	0	225°
1	0	315°

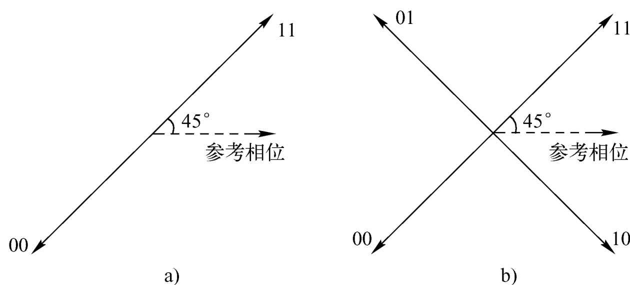


图 4−2 QPSK 信号矢量图

a) BSPK b) QPSK

对于 BPSK，可以看成是 QPSK 的简化，即每个码元含有 1 比特信息，在表 4−1 中，ab 仅采用其中的两种组合，即 00 和 11。

由于 BPSK 是 QPSK 的简化，所以调制器可以使用同一种方案来实现。调制器的方框图如图 4−3 所示。图中，输入的串行二进制信息序列经过串－并转换，分成两路速率减半的序列，再由脉冲发生器产生双极性二电平脉冲信号 $I(t)$和 $Q(t)$，映射规则为“1”→“+1”，“0” →“−1”。然后对 $\cos\omega_c t$ 和 $\sin\omega_c t$ 进行调制，相加后便得到如图 4−2 所示的 BPSK 或 QPSK 信号。不过，对于 BPSK 信号要注意一点，其串—并转换前后的速率不发生变化。

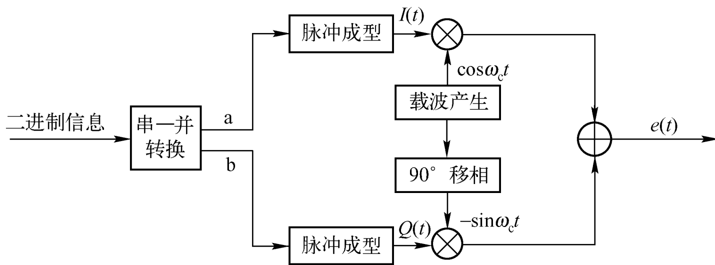


图 4-3 QPSK 信号产生方法

上述的调制方案被称为绝对相移键控，为了提高系统性能，一般解调时会采用相干解调。但是相干解调在载波恢复的时候会出现相位模糊现象。为了解决这个问题，一般会使用差分相移键控，即用前后码元的相位差来表示数字信息。

四进制 DPSK 通常记为 QDPSK。参考图 4-4 所示的 DQPSK 信号编码的规则，可以写出 DQPSK 信号编码规则，如表 4-2 所示。表中$\Delta\varphi_n$是相对于前一相邻码元初相的相位变化。当然，对于 DBPSK，只需使用两种载波变化就可以了，即 0° 和 180°。

表 4-2 DQPSK 信号载波相位编码逻辑关系

双比特码元		载波相位变化（$\Delta\varphi_n$）
a	b	
0	0	0°
0	1	90°
1	1	180°
1	0	−90°

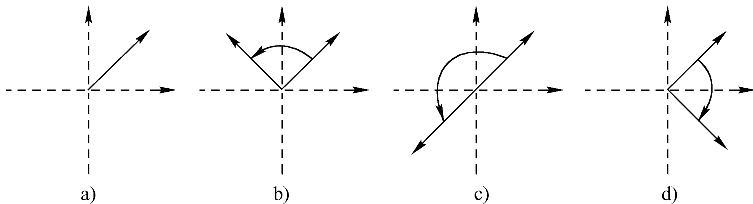


图 4-4 DQPSK 信号矢量图

a) $\Delta\varphi$=0° b) $\Delta\varphi$=90° c) $\Delta\varphi$=180° d) $\Delta\varphi$=−90°

4.2.2 脉冲成型的设计

在上述的调制方案中，I、Q 支路的脉冲成型是一样的，为了简化分析，我们将仅讨论一个支路。不失一般性，以 I 支路为例来分析。

I 支路的信号可以表示为

$$e_{\mathrm{I}}(t)=I(t)\cos\omega_{\mathrm{c}}t=\left(\sum_{n}a_n g(t-nT_{\mathrm{s}})\right)\cos\omega_{\mathrm{c}}t \tag{4-9}$$

其中，a_n 就是被调制的信息，为了简化分析，可以令其取值为+1 或−1。$g(t)$ 是一段模拟信号，它与 a_n、$\cos\omega_{\mathrm{c}}t$ 相乘，就完成了调制。$g(t)$ 的波形设计，就是这部分内容要重点讨论

的脉冲成型问题。

脉冲成型设计，即 $g(t)$ 的波形设计的基本要求就是要保证调制信息不能丢失。要满足这样的要求，把 $g(t)$ 设计成矩形脉冲最简单了。这样脉冲成型设计的结果如图 4−5 所示。

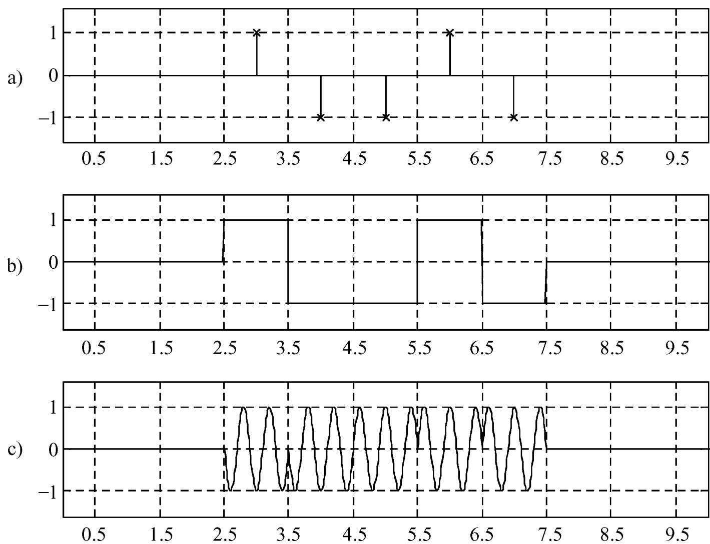


图 4−5　矩形脉冲成型设计

a) 数字信号波形　b) 脉冲成型后的波形　c) 调制后的波形

这样的设计存在一个严重的问题，就是在信号相位翻转的位置调制波形出现了不连续的跳变。比如横轴的 3.5、5.5 和 6.5 这些位置，就出现了急剧的跳变。这样的跳变带来的问题就是信号带宽的展宽，而展宽后的信号通过带限信道就会有频谱的损失，进而令解调后的基带波形出现严重的失真和码间串扰。

为了克服这个问题，奈奎斯特指出，只要把通信系统（包括发射机、信道和接收机）的整个响应设计成在接收端的每个采样时刻只对当前的符号有响应，而对其他符号的响应全为零，那么码间串扰的影响就能被完全抵消。

根据奈奎斯特准则，人们一般把脉冲成型滤波器设计成余弦滚降滤波器。余弦滚降滤波器的传递函数是

$$H(f)=\begin{cases} T_\mathrm{b} & 0\text{剟}|f| \quad \dfrac{1-\alpha}{2T_\mathrm{b}} \\ \dfrac{T_\mathrm{b}}{2}\left\{1+\cos\left[\dfrac{\pi T_\mathrm{b}}{\alpha}\left(|f|-\dfrac{1-\alpha}{2T_\mathrm{b}}\right)\right]\right\} & \dfrac{1-\alpha}{2T_\mathrm{b}}\text{剟}|f| \quad \dfrac{1+\alpha}{2T_\mathrm{b}} \\ 0 & |f|\ldots\dfrac{1+\alpha}{2T_\mathrm{b}} \end{cases} \tag{4−10}$$

其中，α是滚降因子，取值范围是 0～1；T_b是符号周期。与之相对应的滤波器的冲激响应是

$$h(t)=\frac{\sin\left(\pi t/T_\mathrm{b}\right)}{\pi t/T_\mathrm{b}}\cdot\frac{\cos\pi\alpha t/T_\mathrm{b}}{1-4\alpha^2t^2/T_\mathrm{b}{}^2} \tag{4−11}$$

升余弦滚降系统如图 4-6 所示。其中，W_1为理想低通带宽。

图 4-6　升余弦滚降系统

a) 传输函数　b) 冲激响应

我们再看这样的脉冲成型设计的波形如图 4-7 所示。

图 4-7　奈奎斯特脉冲成型设计

a) 数字信号波形　b) 脉冲成型后的波形　c) 调制后的波形

经过这样的设计，可以看出在信号相位翻转的位置，调制波形虽然相位不连续，但是由于载波在此处的幅度很小，使得不连续的跳变就非常不明显了。也就是说，这样设计的系统的带宽的确变窄了，经过带限信道不会对系统的频谱造成太大的损失。当然，这也可以从图 4-6a 中的传输函数看出来。

虽然其冲激响应理论上是无限长的，但是其他符号在当前符号的最大值点的贡献是 0，因此没有码间串扰。但是，请大家注意，这样的无码间串扰是有条件的，即解调端的采样点必须严格地落在每个符号的中间。否则仍然会产生码间串扰。

为此，本节介绍的调制器所采用的脉冲成型设计采用了一种特殊的设计。其基本规则就是如果发送的数字信号是连 1 码或连 0 码，脉冲成型电路输出+1 或-1，当前后码元不一样时，从前一码元的中间点开始到后一码元的中间点之间用正弦或余弦函数来过渡，其过渡点要为 0，而且这当中只能单调下降或上升一次。采用这样的方式的脉冲成型设计如图 4-8 所示。

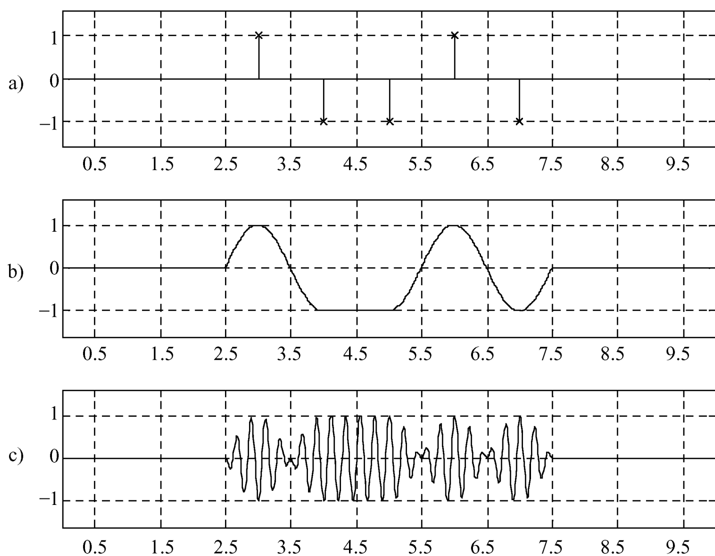


图 4-8 正弦脉冲成型设计

a) 数字信号波形 b) 脉冲成型后的波形 c) 调制后的波形

从图 4-8 可以看出，经过这样的设计，在信号相位翻转的位置调制波形虽然相位不连续，但是由于载波在此处的幅度很小，使得不连续的跳变就非常不明显了。这与奈奎斯特波形成型的结果是一样的。也就是说，这样设计的系统的带宽会很窄，经过带限信道后也就不会令系统的频谱造成太大的损失。接收端的基带波形失真也会小很多。

同时，这样的设计使一个符号的信号波形被限制在了一个符号周期之内，不会拓展到其他符号周期内，自然也就没有了码间串扰的问题。这样的设计还有一个额外的好处，就是接收端采样点不用像奈奎斯特成形方式那样，必须严格对准符号的中点。只要在符号周期内找一个信号比较大的点采样就可了。

4.2.3 调制器的 MATLAB 仿真

为了便于读者理解调制的过程，我们先用 MATLAB 来实现调制器。

1. 生成调制符号

g_bincode 是输入的数字信号序列，每两个数字信号生成一个 DQPSK 调制符号。为了方便起见，增加两个符号，一个是开载波符号，另一个是关载波符号。

【例 4-1】 用MATLAB实现调制器。

```
%符号列表：11=0=>0°；01=1=>+90°；00=2=>180°；10=3=>−90°；4=>关载波；5=>开载波；
symb_size=floor( length(g_bincode)/2);%符号长度原始符号前加入符号 5,结尾加入符号 4
%生成 QPSK 的符号
g_symbol=[5];%第一个符号是开载波
for i=1:symb_size
    tcode=g_bincode(i*2−1)*2+g_bincode(i*2);
    switch tcode
        case 0
            tsym=2;
```

```
            case 1
                tsym=1;
            case 2
                tsym=3;
            case 3
                tsym=0;
        end
        g_symbol=[g_symbol tsym];
    end
    g_symbol=[g_symbol 4]; %最后一个符号是关载波
```

2. 产生基带波形

因为实现的 DQPSK，每个输入的符号代表的是相对相位。为此，使用 curphase 表示前一符号的绝对相位，这样根据当前输入的符号，就可以知道当前符号对应的绝对相位。

为了实现相邻符号之间基带波形的平滑过渡。基带信号的某一个支路的波形有如下 8 种情况：

- 保持 0 电平。
- 保持+1 电平。
- 保持−1 电平。
- 从 0 过渡到+1，对应开载波。
- 从−1 过渡到 0，对应关载波。
- 从+1 过渡到 0，对应关载波。
- 从+1 过渡到−1。
- 从−1 过渡到+1。

【例 4−2】 用MATLAB产生基带波形。

```
%生成基带信号
[g_basei,g_baseq]=basegen(5);
for i=2:length(g_symbol)
    [ti,tq]=basegen(g_symbol(i));
    g_basei=[g_basei ti];
    g_baseq=[g_baseq tq];
end
```

其中，basegen 函数的代码如下：

```
%i,q 代表当前符号周期内的波形序号
function[basei,baseq]=basegen(insymbol)
persistent curphase;
%curphase 记录当前符号的绝对相位，等下一符号到达，就自然记录了上一符号的绝对相位
%绝对相位的定义:0=45°；1=135°；2=225°；3=315°；4=无载波

if insymbol==5
    i=3;q=3;%开载波，绝对相位总是 45°
    curphase=0;
else
```

```
switch insymbol
    case 0 %相对相移 0°
        switch curphase
            case 0
                i=1;q=1;curphase=0;
            case 1
                i=2;q=1;curphase=1;
            case 2
                i=2;q=2;curphase=2;
            case 3
                i=1;q=2;curphase=3;
            case 4
                i=0;q=0;curphase=4;
        end
    case 1 %相对相移 90°
        switch curphase
            case 0
                i=6;q=1;curphase=1;
            case 1
                i=2;q=6;curphase=2;
            case 2
                i=7;q=2;curphase=3;
            case 3
                i=1;q=7;curphase=0;
            case 4
                i=3;q=3;curphase=0;
        end
    case 2 %相对相移 180°
        switch curphase
            case 0
                i=6;q=6;curphase=2;
            case 1
                i=7;q=6;curphase=3;
            case 2
                i=7;q=7;curphase=0;
            case 3
                i=6;q=7;curphase=1;
            case 4
                i=3;q=3;curphase=0;
        end
    case 3 %相对相移−90°
        switch curphase
            case 0
                i=1;q=6;curphase=3;
            case 1
                i=7;q=1;curphase=0;
            case 2
                i=2;q=7;curphase=1;
```

```
                case 3
                    i=6;q=2;curphase=2;
                case 4
                    i=3;q=3;curphase=0;
            end
        case 4 %关闭载波
            switch curphase
                case 0
                    i=4;q=4;curphase=4;
                case 1
                    i=5;q=4;curphase=4;
                case 2
                    i=5;q=5;curphase=4;
                case 3
                    i=4;q=5;curphase=4;
                case 4
                    i=0;q=0;curphase=4;
            end
    end
end
%下面根据 i、q 来生成该符号周期内的两路基带信号
switch i
    case 0
        basei=zeros(1,256);
    case 1
        basei=ones(1,256);
    case 2
        basei=-ones(1,256);
    case 3
        basei=[zeros(1,128) sin(pi/256:pi/256:pi/2)];
    case 4
        basei=[cos(pi/256:pi/256:pi/2) zeros(1,128)];
    case 5
        basei=[-cos(pi/256:pi/256:pi/2) zeros(1,128)];
    case 6
        basei=cos(pi/256:pi/256:pi);
    case 7
        basei=-cos(pi/256:pi/256:pi);
end
switch q
    case 0
        baseq=zeros(1,256);
    case 1
        baseq=ones(1,256);
    case 2
        baseq=-ones(1,256);
    case 3
```

```
        baseq=[zeros(1,128) sin(pi/256:pi/256:pi/2)];
    case 4
        baseq=[cos(pi/256:pi/256:pi/2) zeros(1,128)];
    case 5
        baseq=[−cos(pi/256:pi/256:pi/2) zeros(1,128)];
    case 6
        baseq=cos(pi/256:pi/256:pi);
    case 7
        baseq=−cos(pi/256:pi/256:pi);
end
```

3. 调制的实现

最后的调制比较简单，具体的实现代码如下：

```
%调制
fc=1500;%载波频率
phaseinc=2*pi*fc/8000;%采样间隔内的载波相位增量
fc_i=cos(0:phaseinc:phaseinc*(length(g_basei)−1));
g_modi=fc_i .* g_basei;%i 支路已调信号
fc_q=sin(0:phaseinc:phaseinc*(length(g_basei)−1));
g_modq=fc_q .* g_baseq;%q 支路已调信号
g_mod=g_modi + g_modq;%完整的已调信号
```

4.2.4 调制器的 dsPIC 实现

调制器的相关参数如下：

- 采样速率：8000 样点/s。
- 符号速率：31.25 符号/s。
- 载波频率：1500Hz。
- 信号带宽：31Hz 左右。

为了减少系统的运算量，脉冲成型、载波生成都采用了查表法来实现。调制器的整体框图如图 4−9 所示。

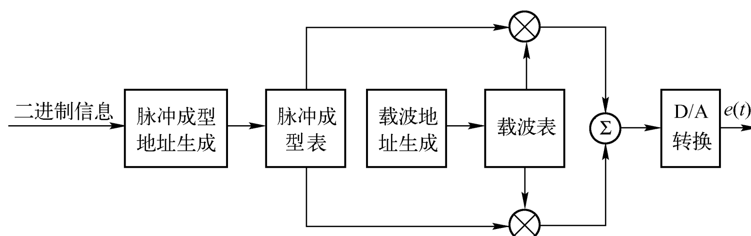


图 4−9　调制器的 dsPIC 实现框图

因为系统的采样率是 8000 样点/s，所以在 dsPIC 系统上首先设计一个定时器，其周期是 1/8000 s ，这样在每个定时中断内计算一次调制器的输出值。

1. 脉冲成型

信号的采样率是 8000 样点/s，符号速率是 31.25 符号/s，因此每个符号周期内共有

8000/31.25=256 个采样点。脉冲成型表共有 2048 个采样点，即占用 8 个符号宽度。其代码如下：

```
const float PSKShapeTbl[2048]=
{
0.0,
0.0,
…
0.999698819,
0.999924702
}
```

为了便于理解系统的原理，把这个数据表用 MATLAB 画出图形来，其图形如图 4-10 所示。

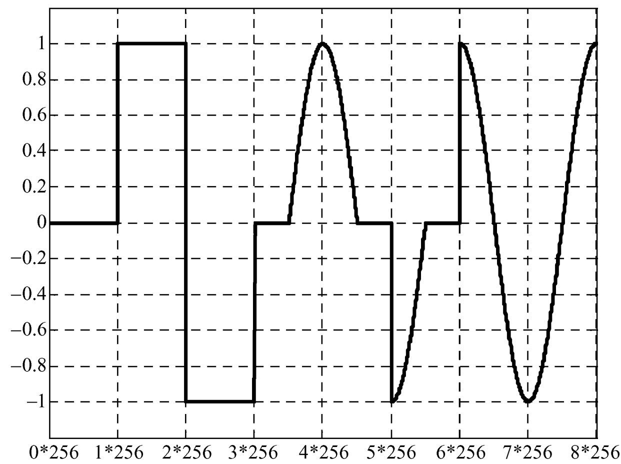


图 4-10　调制器的脉冲成型表

为了便于分析，我们把上面的数据分为 8 组，如表 4-3 所示。

表 4-3　波形成型表

序　　号	波　　形	序　　号	波　　形
0		4	
1		5	
2		6	
3		7	

系统每输入一个符号的信息（BPSK 是 1 比特，QPSK 是 2 比特），脉冲成型电路就输出上表中的某一段数据，共 256 个采样点。但是，要注意的是，上面的表中每段数据并不是正好对应着一个符号，而是对应了前一个符号的后半段和当前符号的前半段。

之所以要这样设计是基于两点考虑。其一是要在波形成型的同时完成差分编码；其二是因为这个数字调制器的波形成型是比较特殊的，当前的信号波形从时间上看虽然被限制在了一个符号周期内，但是其波形形状是跟前、后两个符号的形状相关的。

如果当前符号与前一符号的极性相同，则这个符号的前半周期是+1 或−1，如果与前一符号的极性相反，则这个符号的前半周期是以 0 开始，以+1 或−1 结束的 1/4 周期的正弦波。当前符号的后半周期的情况是类似的，其形状是与后一符号的极性相关的。如果当前符号与后一符号的极性相同，则这个符号的后半周期是+1 或−1，如果与后一符号的极性相反，则这个符号的后半周期是以+1 或−1 开始，以 0 结束的 1/4 周期的余弦波。

从图 4−2b 可以看出，调制系统共有 4 种可能的相位，但是调制器不是一直工作，即有数据要发送的时候才开始发送调制后的载波，如果没有数据要发送，就应该关闭载波的发送。这样做当然可以节约能源，但是更重要的是在时分双工系统中，由于收发双方共用一个载波频点，收发双方任何一个调制器如果始终发送信号，那么系统将无法完成双向通信。这样，除了上述的 4 种相位状态外，还应该加上一个 0 载波，或者无信号（载波关闭）状态。我们给这 5 个状态分别分配一个数字，如图 4−11 所示。

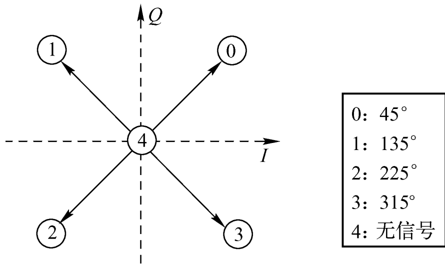


图 4−11　绝对相位的对应关系

为了实现差分编码，我们要根据输入的信息比特计算出需要的相位变化值。相位变化状态可以有以下 4 种：相位不变（增加 0°），增加 90°，增加 180°，减少 90°。为了实现载波的关闭和打开，我们还需要再增加两个变化状态：载波关闭，载波打开。

我们用下面的数组来实现脉冲波形地址的生成。

```
const int PSKPhaseLookupTable [6][5] =
{
                        /*  前一符号绝对相位  当前符号绝对相位  I 支路波形序号  Q 支路波形序号*/
/*相位不变（增加 0°）  */
0x110,     /*     45°            45°          1          1      */
0x211,     /*     135°           135°         2          1      */
0x222,     /*     225°           225°         2          2      */
0x123,     /*     315°           315°         1          2      */
0x004,     /*    无信号          无信号        0          0      */
```

```
/*增加 90°    */
0x611,    /*    45°     135°    6    1    */
0x262,    /*    135°    225°    2    6    */
0x723,    /*    225°    315°    7    2    */
0x170,    /*    315°    45°     1    7    */
0x330,    /*    无信号   45°     3    3    */
/*增加 180°    */
0x662,    /*    45°     225°    6    6    */
0x763,    /*    135°    315°    7    6    */
0x770,    /*    225°    45°     7    7    */
0x671,    /*    315°    135°    6    7    */
0x330,    /*    无信号   45°     3    3    */
/*减少 90°    */
0x163,    /*    45°     315°    1    6    */
0x710,    /*    135°    45°     7    1    */
0x271,    /*    225°    135°    2    7    */
0x622,    /*    315°    225°    6    2    */
0x330,    /*    无信号   45°     3    3    */
/*  载波关闭*/
0x444,    /*    45°     无信号   4    4    */
0x544,    /*    135°    无信号   5    4    */
0x554,    /*    225°    无信号   5    5    */
0x454,    /*    315°    无信号   4    5    */
0x004,    /*    无信号   无信号   0    0    */
/*  载波打开  */
0x110,    /*    45°     45°     1    1    */
0x710,    /*    135°    45°     7    1    */
0x770,    /*    225°    45°     7    7    */
0x170,    /*    315°    45°     1    7    */
0x330     /*    无信号   45°     3    3    */
};
```

上述的数组由 6 组数字组成，每组有 5 个整数，共 30 个整数。每组内数字的位置代表了前一符号的绝对相位。每个整数使用其低 24 比特。比特 7～0 表示当前符号的绝对相位，比特 15～8 表示 Q 支路信号波形在波形表中的序号，比特 23～16 表示 I 支路信号波形在波形表中的序号。

使用数组 PSKPhaseLookupTable [6][5]就可以方便地实现 QPSK 的调制了。我们用一个例子来说明这个数组的使用。假设要被发送的一组二进制信息是：01110010。调制器的载波刚开始处于关闭状态，而且发送完毕后还需要把载波关闭。这样，整个调制过程说明如下。

这一组信息需要 4 个符号周期来发送出去，因为调制载波刚开始处于关闭状态，因此还需要两个额外的符号周期，发送之前需要一个符号周期把载波打开，发送结束后需要一个符号周期把载波关闭。对于发送前多出的一个符号，或可这样理解，因为是差分调制系统，需要一个额外的符号来给出最初的绝对相位，这样才能用载波相位差传递信息。下面我们分 6 步来说明整个调制过程。

1）因为载波关闭对应图 4−11 中的 4 号相位（无信号），第一个输入的符号是“载波打开”，所以选用数组 PSKPhaseLookupTable 中的 PSKPhaseLookupTable [5][4]= 0x330，用它来生成基带波形。这个数字告诉我们：当前绝对相位是图 4−11 中 0 号相位（45°），I、Q 支路的波形都选用表 4−3 中的 3 号波形。

2）前一符号的绝对相位是图 4−11 中的 0 号相位（45°），输入的数字信息是 01，根据表 4−2，需要发送的符号是“增加 90°”，所以选用数组 PSKPhaseLookupTable 中的 PSKPhaseLookupTable [1][0]= 0x611，用它来生成基带波形。这个数字告诉我们：当前绝对相位是图 4−11 中 1 号相位（135°），I、Q 支路的波形分别选用表 4−3 中的 6 号和 1 号波形。

3）前一符号的绝对相位是图 4−11 中的 1 号相位（135°），输入的数字信息是 11，根据表 4−2，需要发送的符号是“增加 180°”，所以选用数组 PSKPhaseLookupTable 中的 PSKPhaseLookupTable [2][1]= 0x763，用它来生成基带波形。这个数字告诉我们：当前绝对相位是图 4−11 中的 3 号相位（315°），I、Q 支路的波形分别选用表 4−3 中的 7 号和 6 号波形。

4）前一符号的绝对相位是图 4−11 中的 3 号相位（315°），输入的数字信息是 00，根据表 4−2，需要发送的符号是“相位不变”，所以选用数组 PSKPhaseLookupTable 中的 PSKPhaseLookupTable [0][3]=0x123，用它来生成基带波形。这个数字告诉我们：当前绝对相位是图 4−11 中的 3 号相位（315°），I、Q 支路的波形分别选用表 4−3 中的 1 号和 2 号波形。

5）前一符号的绝对相位是图 4−11 中的 3 号相位（315°），输入的数字信息是 10，根据表 4−2，需要发送的符号是“减少 90°”，所以选用数组 PSKPhaseLookupTable 中的 PSKPhaseLookupTable [3][3]=0x622，用它来生成基带波形。这个数字告诉我们：当前绝对相位是图 4−11 中的 2 号相位（225°），I、Q 支路的波形分别选用表 4−3 中的 6 号和 2 号波形。

6）前一符号的绝对相位是图 4−11 中的 2 号相位（225°），需要发送的符号是“载波关闭”，所以选用数组 PSKPhaseLookupTable 中的 PSKPhaseLookupTable [4][2]=0x554，用它来生成基带波形。这个数字告诉我们：当前绝对相位是图 4−11 中的 4 号相位（无信号），I、Q 支路的波形都选用表 4−3 中的 5 号波形。

至此，这个调制过程就结束了。把这个过程用图 4−12 来表示。

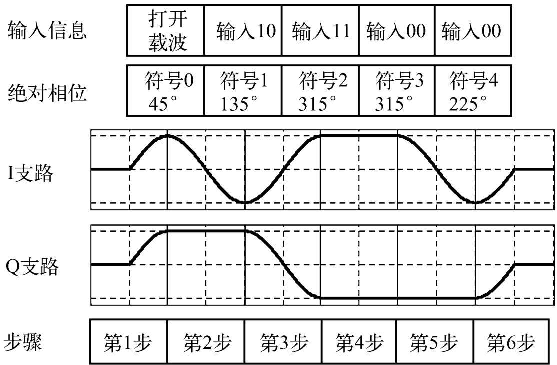


图 4−12　调制脉冲成型示例

需要再次说明的是，图 4-12 中，每一步的每个支路都要输出 256 个采样值，每一步占用的时间是 1/31.25s，这样输出信号的采样率才能是 8000 样点/s。再联系到调制器的以 1/8000s 为周期的定时器，实际的操作就是每 256 个定时中断计算一次需要输出的符号，然后在下面的 256 个定时中断中，每次输出一个采样点数据。

2. 载波产生

为了简化运算和提高计算速度，载波的生成也采用查表法。这个数据表的代码大致如下：

```
const   int Sine_table [ 1025 ] = {
        0x0000,
        0x0032,
        0x0064,
        0x0096,
……
        0x7FFE,
        0x7FFE,
        0x7FFE,
        0x7FFF
};
```

把这个表用 MATLAB 画出来，如图所示 4-13。

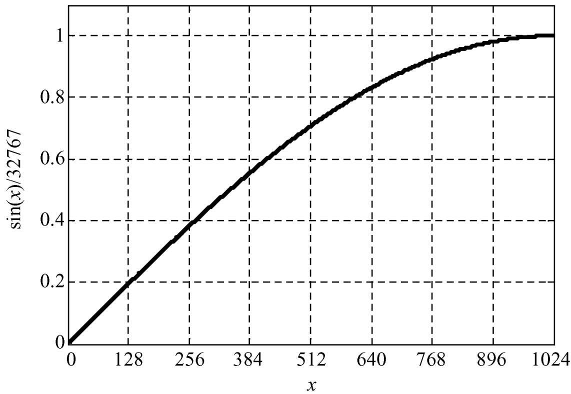


图 4-13　载波的生成数据

这些数据实际上是 1/4 周期正弦波的数据。这样是为了减少存储空间，因为一个正弦波的周期内的其他部分的形状与第一个 1/4 周期内的部分是基本一样的，在使用载波时，只要稍微用一点小技巧，就可得到一个周期内的几乎任何位置的数据。这个处理函数如下：

```
int    Sine_lookup (    int phase ) {

        int quadrant, angle, result;

        quadrant = phase / 1024;//计算相位在哪个 1/4 周期内
        angle = phase % 1024;
```

```
        switch ( quadrant ) {
            case 0://在第 0 个 1/4 周期内，直接输出表内数据
                result = Sine_table [ angle ];
                break;
            case 1: //在第 1 个 1/4 周期内，倒置输出表内数据
                result = Sine_table [ 1024 - angle ];
                break;
            case 2: //在第 2 个 1/4 周期内，将表内数据乘以-1 后输出
                result = -Sine_table [ angle ];
                break;
            case 3: //在第 3 个 1/4 周期内，将表内数据倒置并乘以-1 后输出
                result = -Sine_table [ 1024 - angle ];
                break;
        }
        return result;
    }
```

有了这个函数只要知道相位值（phase），就可以由此推算出载波的数值了。因为我们的数据表不可能是无限精细的，因此相位值只能是 $\frac{2\pi/4}{1024}=\frac{\pi}{2048}\approx 0.088^{\circ}$ 的整数倍，即 *phase* 的取值范围是 0～4095。这样的设计虽然没有实现绝对的准确，但是也能满足要求了。

之所以这样设计数据表，是因为调制器的载波频率很难与 8000 形成整数倍的关系，因此，每个采样点在一个载波周期内的位置不会是一个很完美的位置，相邻周期内，采样点的位置肯定也不相同。所以，数据表内的取样间隔只有尽量的密，才能保证精度要求。

下面说明一下如何计算每个采样点位置的载波相位。计算公式如下：

$$phase=\frac{TXPhase}{\Delta\varphi_n} \tag{4-12}$$

其中，$\Delta\varphi_n=\frac{2\pi}{4096}\approx\frac{1}{651.8986469}$ 是载波数据表内相邻采样值的相位差；*TXPhase* 是当前载波在采样（采样间隔是 1/8000s）点的相位；*phase* 是整数，在使用函数 Sine_lookup(int phase)之前，需要对 4096 取余数。

这样 I、Q 支路载波的计算代码如下：

```
    TXPhase += (long) (KCONV*m_PSKPhaseInc);
        TX_I = (float)(Sine_lookup ((TXPhase >> 4 ) & 0x0FFF));
        TX_Q = (float)(Sine_lookup (((TXPhase >> 4 ) + 0x400) & 0x0FFF));
```

其中，m_PSKPhaseInc 是载波在 1/8000s 的采样间隔内的相位增量。用下面的公式计算：

$$m_PSKPhaseInc=\frac{2\pi\times m_TxFreq}{8000} \tag{4-13}$$

式中，m_TxFreq 是载波频率。

代码中 KCONV=651.8986469×16，也就是把真实的相位值扩大了 16 倍，在后面的代

码中用右移 4 位来修正。这样做是为了提高精度，防止直接相乘取整后的四舍五入降低系统精度。

Q 支路的相位值就直接在 I 支路的相位上增加了 0x400，这其实正好对应着π/4，即把余弦函数转换为正弦函数。

这样得到的 TX_I 和 TX_Q 就是载波的数值了，但是数值的取值范围不是一般的 0～1 之间，而是放大了 32767 倍，即 0～32767。

下面只要在每个定时中断（定时间隔是 1/8000 s ）中计算一下 I、Q 支路的载波与基带脉冲采样的对应点的乘积并相加就可以得到调制后的信号。不过这时的信号还是数字信号，通过一个 D/A 转换器件，就得到了真正的模拟的被调制到音频载波的 QPSK 调制信号了。

4.3 解调

由于调制器采用的是差分调制器，因此解调器也必须采用差分解调器。实现差分解调器基本上有两大类方法：相干解调加逆码变换和相位比较法。因为前一种方法在相同信噪比的情况下，可得到更低的误码率，因此在信道质量不好的场合得到了广泛的应用。下面介绍的解调器也将采用相干解调加逆码变换的方式实现。

4.3.1 解调器的基本原理

把频带数字信号还原为基带数字信号的过程称为数字解调。DQPSK 解调原理图如图 4-14 所示。采用相干解调方法，即用两路正交的相干载波，可以很容易地分离出这两路正交的基带信号，解调后的两路基带信号判决后就得到了 I、Q 两路的码元 c 和 d，经过逆码变换和并/串转换后，成为串行数据输出。

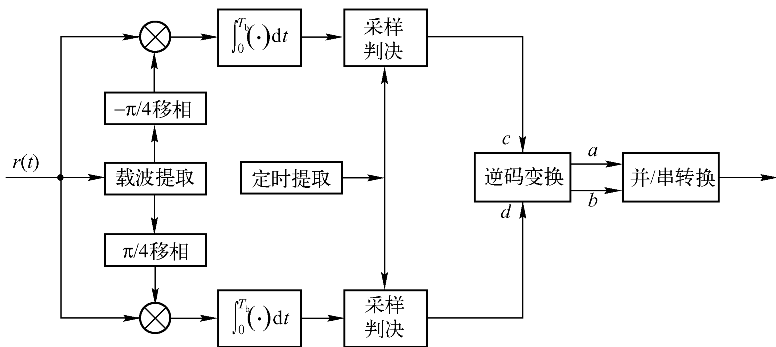


图 4-14 解调器的基本原理

不考虑噪声及传输畸变时，输入到解调器的 QPSK 信号码元可表示为

$$r(t) = a\cos(\omega_c t + \varphi_n) \tag{4-14}$$

上、下两支路相乘器输出分别为

$$\begin{cases} z_A(t)=a\cos(\omega_c t+\phi_n)\cos\omega_c t=\dfrac{a}{2}\cos(2\omega_c t+\phi_n)+\dfrac{a}{2}\cos\phi_n \\ z_B(t)=a\cos(\omega_c t+\varphi_n)\left[-\sin\omega_c t\right]=\dfrac{-a}{2}\sin(2\omega_c t+\varphi_n)+\dfrac{a}{2}\sin\varphi_n \end{cases} \tag{4-15}$$

从解调直接的输出可以看出每个支路都有两项，其中 $\frac{a}{2}\cos(2\omega_c t+\phi_n)$ 的中心频率是载频的两倍，而 $\frac{a}{2}\cos\phi_n$ 则就是纯粹的基带信号，也就是我们需要的部分。面对这样的情况，我们可以容易地使用低通滤波器来抑制掉高频部分，而保留基带信号为后续处理做准备。图 4−14 中相乘器后面是一个积分器，应该说积分器也是低通滤波器的一种。

积分器或者说低通滤波器的输出（省略幅度成分）就可以写成

$$\begin{cases} x_A(t)=\cos\varphi_n \\ x_B(t)=\sin\varphi_n \end{cases} \tag{4-16}$$

按照 φ_n 的取值不同，此电压可能为正，也可能为负，故是双极性电压。在编码时曾经规定：“0”→“−1”，“1” →“+1”，因此判决时，也把正电压判为“1”，负电压判为“0”。因此得出判决规则如表 4−4 所示。

表 4−4　相干正交解调的判决准则

信号码元相位	I 支路输出	Q 支路输出	判决器输出	
			c	d
45°	+	+	1	1
135°	−	+	0	1
225°	−	−	0	0
315°	+	−	1	0

逆码变换器的功能与发送端的相反，它需要将判决器输出的相对码恢复成绝对码。设逆码变换器当前的输入数据为 c_n、d_n，前一时刻输入数据为 c_{n-1}、d_{n-1}，输出数据为 a_n、b_n。现在我们来举例说明它是如何完成所要求的功能的。假设输入解调器的信号相位序列为

$\{\varphi_n\}$：45° 135° 135° 315° 225° 45° 315° …

按表 4−4 可知，能得到该相位的逆码变换器输入数据序列 $\{c_n \quad d_n\}$ 为

$\{c_n \quad d_n\}$：11 01 01 10 00 11 10 …

那么前后码元相位差为

$\{\Delta\varphi_n\}$：90° 0° 180° 270° 180° 270° …

按表 4−2 的规定，可以得到应变换成的绝对码为

$\{a_n \quad b_n\}$：01 00 11 10 11 10 …

即 $c_n\ d_n$ 由“11”变为“01”，表明相位变化了 90°，因此逆码变换器输出应为“01”，再由“01”变为“01”，说明相位保持没变，因此码逆变换结果应为“00”，依此类推得上述结果。

为了正确地进行逆码变换，这些码元之间的关系应该符合表 4−2 中的规则。为此把表 4−4 中的各行按 $c_{n-1}d_{n-1}$ 的组合为序重新排列，构成表 4−5。从这个表中可以找出，由逆码变换器的当前输入 $c_n d_n$ 和前一时刻的输入 $c_{n-1}d_{n-1}$，得到逆码变换器当前输出 $a_n b_n$ 的规律。

表 4-5　QDPSK 逆码变换器的逻辑功能

前一时刻输入			本时刻输入			输出数据	
φ_{n-1}	c_{n-1}	d_{n-1}	c_n	d_n	$\Delta\varphi_n$	a_n	b_n
45°	1	1	1 0 0 1	1 1 0 0	0° 90° 180° 270°	0 0 1 1	0 1 1 0
135°	0	1	1 0 0 1	1 1 0 0	270° 0° 90° 180°	1 0 0 1	0 0 1 1
225°	0	0	1 0 0 1	1 1 0 0	180° 270° 0° 90°	1 1 0 0	1 0 0 1
315°	1	0	1 0 0 1	1 1 0 0	90° 180° 270° 0°	0 1 1 1	1 1 0 0

4.3.2　解调器的 MATLAB 仿真

【例 4-3】 用MATLAB仿真QDPSK的解调过程。

解调器的总体构架如下：

```
for i=1:length(g_mod)
        与载波相乘
        低通滤波
        载波同步和符号同步
        差分相位解码
end
```

g_mod 是已调信号。上述构架中载波同步和符号同步会在其他章节涉及，这里不再赘述。下面分别介绍各个部分的 MATLAB 仿真。

1. 载波相乘

已调信号与载波相乘的 MATLAB 代码如下：

```
dembase_ai(i)=g_mod(i)*cos(ph0);
dembase_aq(i)=g_mod(i)*sin(ph0);
```

这部分代码比较简单，但是要注意 ph0 代表载波相位，在每次循环中都需要进行修正。这有两个原因：一个是因为伴随着采样间隔，载波的相位会有一个固定步长在增量。另外，还需要对载波进行同步，即需要小范围修正。

2. 低通滤波

已调信号与载波相乘的结果中除了基带信号以外还有二倍频分量，当然经过实际的信道后，信号中也会存在一定的噪声。为了获得比较纯净的基带信号，需要对相乘的结果进行低通滤波。低通滤波的 MATLAB 代码如下：

```
dembase_bi(i)=fir34down(dembase_ai,i);
dembase_bq(i)=fir34down(dembase_aq,i);
```

```
dembase_ci(i)=fir34down(dembase_bi,i);
dembase_cq(i)=fir34down(dembase_bq,i);
```

从代码可以看出，对相乘结果进行了两级低通滤波。这两级滤波使用了相同的滤波器，其 MATLAB 代码如下：

```
%34 阶低通滤波器
function [outdata] = fir34down(indata,index)
Dec4LPCoef = [%滤波器的系数
                32767 *   −0.00021203644,
                32767 *   −0.00070252426,
                32767 *    −0.0016680526,
                32767 *    −0.0031934799,
                32767 *    −0.0051899752,
                32767 *    −0.0072862086,
                32767 *    −0.0087714235,
                32767 *    −0.0086272102,
                32767 *    −0.0056735648,
                32767 *     0.0011784719,
                32767 *       0.01261353,
                32767 *      0.028615709,
                32767 *      0.048280707,
                32767 *      0.069812051,
                32767 *      0.090735013,
                32767 *       0.10830381,
                32767 *       0.12001897,
                32767 *       0.12413413,
                32767 *       0.12001897,
                32767 *       0.10830381,
                32767 *      0.090735013,
                32767 *      0.069812051,
                32767 *      0.048280707,
                32767 *      0.028615709,
                32767 *       0.01261353,
                32767 *     0.0011784719,
                32767 *    −0.0056735648,
                32767 *    −0.0086272102,
                32767 *    −0.0087714235,
                32767 *    −0.0072862086,
                32767 *    −0.0051899752,
                32767 *    −0.0031934799,
                32767 *    −0.0016680526,
                32767 *   −0.00070252426,
                32767 *   −0.00021203644];

outdata=0;
if index>35
```

```
        for i=1:35
            di=index−i+1;
            outdata=outdata+Dec4LPCoef(i)*indata(di);
        end
    else
        for i=1:index
            di=index−i+1;
            outdata=outdata+Dec4LPCoef(i)*indata(di);
        end
    end
```

3. 差分相位解码

差分相位解码根据前后两个最佳采样点的值计算出前后两个码元的相位差，其代码如下：

```
dltzi=symdemi(symnum)*symdemi(symnum−1) + symdemq(symnum)*symdemq(symnum−1);
dltzq=symdemq(symnum)*symdemi(symnum−1) − symdemi(symnum)*symdemq(symnum−1);
phase=rad2deg(atan2(dltzq,dltzi));
% 符号列表：11=0=>0°；01=1=>+90°；00=2=>180°；10=3=>−90°；4=>关载波；5=>开载波；
if phase>−45 & phase<=45
    g_dembin=[g_dembin 1 1];%0°−>11
elseif phase>45 & phase<=135
    g_dembin=[g_dembin 0 1];%90°−>01
elseif phase>−135 & phase<=−45
    g_dembin=[g_dembin 1 0];%−90°->10
else
    g_dembin=[g_dembin 0 0];%180°->11
end
```

其中，symdemi 是最佳采样点的采样值；phase 是计算出的前后码元的相位差；g_dembin 是解调出的二进制数字信号。

4.3.3 解调器的 dsPIC 实现

解调器的相关参数如下：

- 采样速率：8000 样点/s。
- 符号速率：31.25 符号/s。
- 载波频率：1500Hz。
- 信号带宽：31Hz 左右。

解调器的框图如图 4−15 所示，其采样率在不同位置是不一致的。A/D转换的速率是 8000 采样/s，载波恢复单路的输出也是 8000 采样/s，这样输出的I_1、Q_1信号送到抽取滤波器。整个解调器有两个抽取滤波器，每个抽取滤波器的作用是低通滤波并把抽样速率降低到输入速率的 1/4。这样在抽取滤波器 2 后面的I_2、Q_2处的数据采样速率就成了 500 采样/s。这里的数据被同时送到低通滤波器A和低通滤波器B，前者的输出用来进行AGC计算、载波恢复，后者用来符号同步和取样。符号同步电路从 16 个连续的采样数据找到一个最佳采样点，相位解码就在这个最佳采样点上进行。因此，最终的数据输出的符号速率是 31.25 符号/s。如果是BPSK，数据速率将是 31.25bit/s，如果是QPSK，数据速率将是 62.5 bit/s。

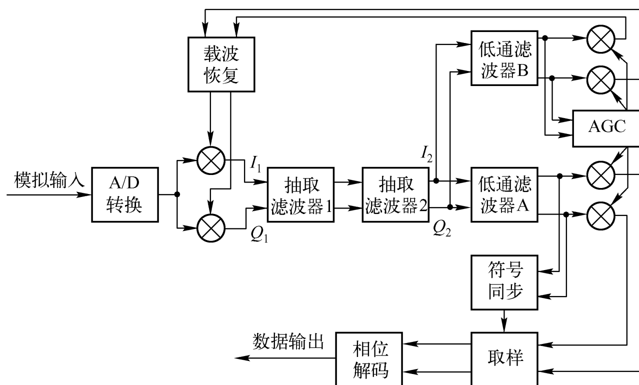


图 4-15　解调器的框图

A/D 转换是在定时器的控制下完成的，定时间隔是 1/8000s，与调制器是一样的。

载波恢复单路的输出与调制器的原理是一样的。唯一不同的是，每个定时间隔内载波的相位增量不一定都是

$$\frac{2\pi \times m_RxFreq}{8000} \tag{4-17}$$

其中，m_RxFreq 是接收信号的载波频率。

因为本地载波的频率、相位与接收信号的载波频率、相位必须严格一致，才能正确地解调。为了跟踪接收信号载波、频率的变化，在必要的时候会适当地对式 4-17 的数值进行修正，以实时地跟踪载波频率、相位的变化。这部分内容会在第 5 章详细讲解。

AGC 部分的功能是对信号的幅度进行控制，它的作用就是使信号在经过 AGC 处理后，幅度基本达到 1 左右。其中，*AGCave* 是对信号平均幅度的估计，让当前的 *I*、*Q* 信号乘以 $\frac{1}{AGCave}$，就可以实现幅度在 1 左右的目的。当然，如果信号本身的平均幅度已经小于 1 了，就不再处理了。

AGC 中比较复杂的是对平均幅度的估计。从图 4-12 可以知道，信号的幅度是有起伏的，而我们希望得到的平均幅度 *AGCave* 最好是图 4-12 中的最大值，为此设计的幅度估计电路如图 4-16 所示

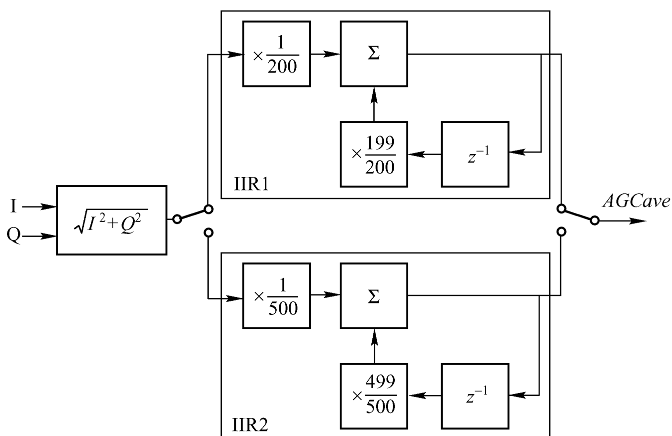


图 4-16　AGC 的幅度估计电路

通过 $r=\sqrt{I^2+Q^2}$ 可以计算出当前信号的幅度。当前信号的幅度比平均幅度高时，说明幅度在增加，此时选用上面的滤波器 IIR1，反之则采用下面的滤波器 IIR2。这两个 IIR 滤波器的 z 变换函数、幅度响应函数如下：

$$H_1(z)=\frac{z/200}{z-199/200} \tag{4-18}$$

$$H_2(z)=\frac{z/500}{z-499/500} \tag{4-19}$$

$$A_1(f)=20\lg\left(\left|\frac{1/200}{\mathrm{e}^{\mathrm{j}\frac{2\pi f}{f_s}}-199/200}\right|\right) \tag{4-20}$$

$$A_2(f)=20\lg\left(\left|\frac{1/500}{\mathrm{e}^{\mathrm{j}\frac{2\pi f}{f_s}}-499/500}\right|\right) \tag{4-21}$$

其中， $f_s=500\text{Hz}$ 。这两个滤波器的阶跃响应和幅频函数如图 4-17 所示。

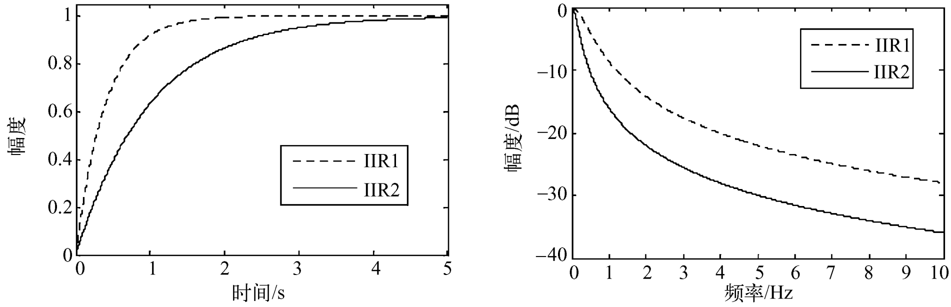


图 4-17　AGC 的 IIR 滤波器的特性

相位解码的输入信号的采样率是 31.25 采样/s，每个采样点的 I、Q 两路信号值就直接代表了当前符号的绝对相位。因为采用的是 QPSK 调制技术，因此，相位解码的关键是计算前后两个符号的相位差。

第 k 个符号的信号可以用下式表示：

$$Z_k=I_k+\mathrm{j}Q_k=A_k\mathrm{e}^{\mathrm{j}\varphi_k}$$

其中， $\varphi_k=\arctan\dfrac{Q_k}{I_k}$

相应的前一个符号，即第 k−1 个符号的信号可以用下式表示：

$$Z_{k-1}=I_{k-1}+\mathrm{j}Q_{k-1}=A_k\mathrm{e}^{\mathrm{j}\varphi_{k-1}}$$

为了计算当前符号与前一符号的相位差，可以使用一个中间变量

$$\begin{aligned}Z&=Z_kZ_{k-1}^*=A_kA_{k-1}\mathrm{e}^{\mathrm{j}(\varphi_k-\varphi_{k-1})}\\&=\left(I_kI_{k-1}+Q_kQ_{k-1}\right)+\mathrm{j}\left(Q_kI_{k-1}-I_kQ_{k-1}\right)\\&=I+\mathrm{j}Q\end{aligned}$$

相应地就可以得到我们所需要的相位差了，即

$$\begin{aligned}\Delta\varphi &= \varphi_k - \varphi_{k-1} \\ &= \arctan\frac{Q_k I_{k-1} - I_k Q_{k-1}}{I_k I_{k-1} + Q_k Q_{k-1}} \\ &= \arctan\frac{Q}{I}\end{aligned}$$

相位差 $\Delta\varphi$ 与 I、Q 的关系还可以用图 4−16a 表示出来。

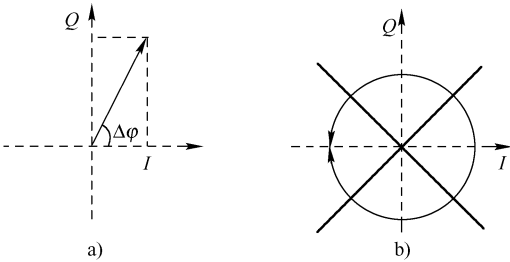


图 4−18　相位的映射关系

显然，$\Delta\varphi$ 的取值范围应该是 0～2π，但是 dsPIC 的 C 软件提供的函数是 $\operatorname{atan}(Q, I)$，这个函数返回值的取值范围是 $-\pi \sim \pi$，如图 4−16b 所示。结合表 4−2，可以进行解码了。

对于 DQPSK，解码的判断原则如表 4−6 所示。

表 4−6　QDPSK 相位解码逻辑

双比特码元		载波相位变化（$\Delta\varphi_n$）
a	b	
0	0	$-45^{\circ} < \Delta\varphi < 45^{\circ}$
0	1	$45^{\circ} < \Delta\varphi < 135^{\circ}$
1	1	$135^{\circ} < \Delta\varphi < 180^{\circ}$ 或 $-180^{\circ} < \Delta\varphi < -135^{\circ}$
1	0	$-135^{\circ} < \Delta\varphi < -45^{\circ}$

对于 DBPSK，解码的判断原则要简单一些。因为 DBPSK 的 $\Delta\varphi$ 只有两种情况，0°和 180°，因此只要判断 I 的正负就可以了。如果 $I>0$，就输出 1，否则就输出 0。

第 5 章　同步功能的设计与实现

同步是通信系统中一个重要的实际问题。当采用同步解调或相干检测时，接收端需要提供一个与发射端调制载波同频同相的相干载波。这个相干载波的获取就称为载波提取，或称为载波同步。

数字通信中，除了有载波同步的问题外，还有位同步的问题。因为消息是一串相继的信号码元的序列，解调时常需知道每个码元的起止时刻。例如在本书前面介绍的解调器结构中，需要对积分器或低通滤波器的输出进行抽样判决。抽样判决的时刻应位于每个码元的终止时刻，因此，接收端产生与接收码元的重复频率和相位一致的定时脉冲序列的过程称为码元同步或位同步，而称这个定时脉冲序列为码元同步脉冲或位同步脉冲。

数字通信中的消息数字流总是用若干码元组成一个“字”，又用若干“字”组成一个“句”。因此，在接收这些数字流时，同样也必须知道这些“字”、“句”的起止时刻。在接收端产生与“字”、“句”起止时刻相一致的定时脉冲序列，称为“字”同步和“句”同步，统称为群同步或帧同步。

当通信是在两点之间进行时，完成了载波同步、位同步和群同步之后，接收端不仅获得了相干载波，而且通信双方的时标关系也解决了。这时，接收端就能以较低的错误概率恢复出数字信息。

同步系统性能的降低，会直接导致通信系统性能的降低，甚至使通信系统不能正常工作。可以说，在同步通信系统中，同步是进行信息传输的前提，正因为如此，为了保证信息的可靠传输，要求同步系统应有更高的可靠性。

本章将分别讨论载波同步、位同步和群同步。

5.1　载波同步

在利用相干解调的系统中，接收端需要一个与发送端同频、同相的载波。以 QPSK 调制为例，假设调制后的信号是 $r(t)=x(t)\cos(\omega_1 t+\varphi_1)$，其中 $x(t)$ 是基带信号，本地载波信号是 $c(t)=\cos(\omega_2 t+\varphi_2)$。这样，在解调器中，本地载波与接收信号相乘后，结果为

$$\begin{aligned} y(t)&=x(t)\cos(\omega_1 t+\varphi_1)\cos(\omega_2 t+\varphi_2)\\ &=\frac{1}{2}x(t)\cos[(\omega_1-\omega_2)t+(\varphi_1-\varphi_2)]+\frac{1}{2}x(t)\cos[(\omega_1+\omega_2)t+(\varphi_1+\varphi_2)] \end{aligned} \tag{5-1}$$

经过低通滤波后，可以得到

$$y'(t)=\frac{1}{2}x(t)\cos[(\omega_1-\omega_2)t+(\varphi_1-\varphi_2)] \tag{5-2}$$

当 $\omega_1=\omega_2$ 并且 $\varphi_1=\varphi_2$ 时，也就是实现了载波同步的情况下，$y'(t)=\frac{1}{2}x(t)$，这样可以不失真地恢复出发送的基带信号 $x(t)$。反之，当 $\omega_1\neq\omega_2$ 或 $\varphi_1\neq\varphi_2$ 时，接收到的 $y'(t)$ 就会出现波形失真、幅度降低等问题。因此发送载波与接收载波必须同频同相。

5.1.1　载波同步的基本原理

提取载波的方法一般分为两类：一类是在发送有用信号的同时，在适当的频率位置上，插入一个（或多个）称为导频的正弦波，接收端就由导频提取载波，这类方法称为插入导频法；另一类是不专门发送导频，而在接收端直接从发送信号中提取载波，这类方法称为直接法。

插入导频必然带来功率的消耗，因此常选用直接法。而直接法中比较经典的是科斯塔斯环，其框图如图 5−1 所示。

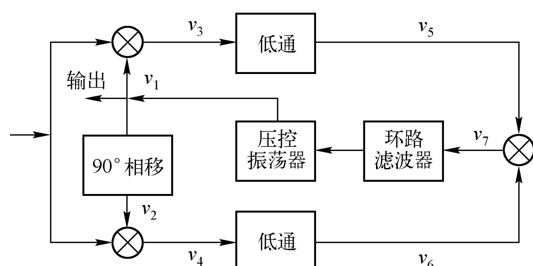


图 5−1　科斯塔斯环框图

加于两个相乘器的本地信号分别为压控振荡器的输出信号 $\cos(\omega_c t+\theta)$ 和它的正交信号 $\sin(\omega_c t+\theta)$。设输入的已调信号为 $m(t)\cos(\omega_c t)$，则

$$v_3=m(t)\cos(\omega_c t)\cos(\omega_c t+\theta)=\frac{1}{2}m(t)[\cos\theta+\cos(2\omega_c t+\theta)] \tag{5-3}$$

$$v_4=m(t)\cos(\omega_c t)\sin(\omega_c t+\theta)=\frac{1}{2}m(t)[\sin\theta+\sin(2\omega_c t+\theta)] \tag{5-4}$$

经低通后的输出分别为

$$v_5=\frac{1}{2}m(t)\cos\theta \tag{5-5}$$

$$v_6=\frac{1}{2}m(t)\sin\theta \tag{5-6}$$

低通滤波器应该允许 $m(t)$ 通过。将 v_5 和 v_6 加于相乘器，得

$$v_7=v_5v_6=\frac{1}{8}m^2(t)\sin 2\theta \tag{5-7}$$

式中，θ 是压控振荡器输出信号与输入已调信号载波之间的相位误差。当 θ 较小时，有

$$v_7\approx\frac{1}{4}m^2(t)\theta \tag{5-8}$$

式（5−8）中 v_7 与相位误差 θ 成正比，它就相当于一个鉴相器的输出。用 v_7 去调整压控振荡器输出信号的相位，最后使稳态相位误差减小到很小的数值。这样压控振荡器输出 v_1 就是

所需提取的载波。

这里有个约束条件，就是$m(t)$的取值只能是+1 或−1，只有这样v_7才能代表载波的相差。

从上面的分析可以看出，科斯塔斯环的本质是通过计算消除了已调信号中的调制信息，而保留了其中的载波相位信息，进而根据这个相差去控制压控振荡器。

5.1.2 载波同步的 MATLAB 实现

【例 5−1】 假定发送信号频率为 1800Hz，本地初始信号频率为 1780Hz，利用科斯塔斯环实现载波同步。利用本地初始频率以及科斯塔斯环，经过多次迭代后，使最后获得的载波频率趋近于实际发送信号的载波频率。程序中直接给出了环路滤波器的参数C1 和C2，具体推导过程可以参见文献[3]，读者可以根据具体的应用对这些参数进行修改。

```
clear
clg
fs=8e3;                 %采样频率
ts=1/fs;                %抽样间隔时间
num=2.5e4;              %仿真的点数
SNR=15;                 %信噪比
real_fc=1800;           %载波频率
data=sin(2*pi*real_fc*(0:num−1)*ts+pi/4)+sqrt(1/(10^(SNR/10)))*randn(1,num);  %产生输入信号

fc=1790;                %本地载波频率

n=fs/100;
nn=[0:n−1];
nf=floor(length(data)/n);
wfc=2*pi*fc;
phi_prv=0;
temp=0;
frame=0;
carrier_phase=0;
phase=0;

c1=6.37;
c2=0.21;

for frame=1:nf
    expcol=exp(j*(wfc*ts*nn+phase));
    sine=imag(expcol);
    cosine=real(expcol);

    x=data((1:n)+((frame−1)*n));

    x_sine=x.*sine;
    x_cosine=x.*cosine;
```

```
        Q=sum(x_sine);
        I=sum(x_cosine);
        phase_discri(frame)=atan(Q/I);

        dfrq=c1*phase_discri(frame)+temp;
        temp=temp+c2*phase_discri(frame);
        wfc=wfc-dfrq*2*pi;
        dfrq_frame(frame)=wfc;
        phase=wfc*ts*n+phase;
    end

    plot(dfrq_frame/(2*pi));
    hold on
    plot([1:length(dfrq_frame)],real_fc,'r');
    legend('锁相环跟踪','实际的载波频率');
    grid
    mean_freq=mean(dfrq_frame/2/pi)
    p=abs(real_fc-mean_freq)/real_fc;
```

运行程序，可以校正本地载波频率至 1800Hz，图 5-2 是本地载波和实际载波之间的关系图，可以发现，经过大约 50 帧后，系统基本就可以锁定了。

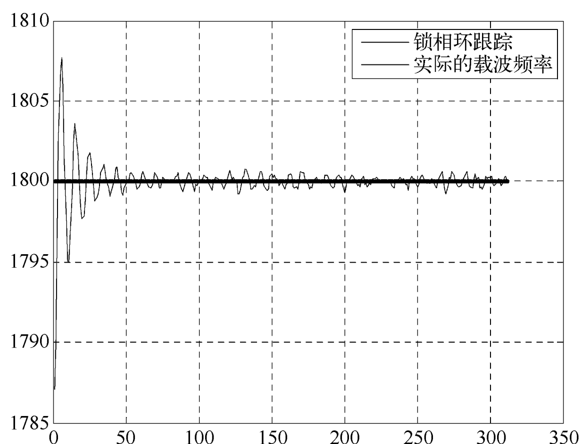


图 5-2　锁相环锁定过程

5.1.3　载波同步的 dsPIC 实现

科斯塔斯环实现的关键是找到本地载波与接收信号载波的相差，由于接收信号被调制过，即每个符号的相位与被调制信息相关，这样为了获得准确的相位差，就需要设法消除信息调制的影响。能否避开这个问题呢？在我们设计的差分调制系统中是可以避开这个问题的。因为差分系统是用相邻符号的相位差来携带调制信息的，因此其载波同步

不需要严格的相位同步，只要本地载波与接收信号的载波同频就可以了，它们之间允许存在一个固定的相差。另一方面，接收信号即便是被调制过的，每个符号内载波的相位是不同的，但是其载波频率是固定的。如果前后符号之间的绝对相位不同，按理说由于相位的突变，频率也会发生突变。但是参考图 4−12，可以看出如果相邻符号的绝对相位发生了变化，那么过渡点的信号幅度会很小，这样对频率估计的影响就可以忽略了。因此，如果仅对频率进行锁定，而允许一个固定相差的存在，那么载波同步的过程中，就不再需要设法消除调制的影响了。

参见图 4−15 中解调器的框图，其中载波恢复部分输出的信号采样率是 8000 采样/s，即恢复的载波信号输出间隔是 $\Delta T=\dfrac{1}{8000}$ s，每个采样信号间隔内载波恢复电路令相位增加一个 $\Delta\varphi$，这样其输出的载波频率就是 $\omega=\dfrac{\Delta\varphi}{\Delta T}$。载波恢复电路的功能就是正确地估计出这个 $\Delta\varphi$，令其输出的载波频率 ω 与接收信号的载波同频。

根据式（5−1）和式（5−2）的分析，图 4−15 中低通滤波器 B 的输出信号可以表示为

$$I(t)=\frac{1}{2}x(t)\cos[(\omega_1-\omega_2)t+(\varphi_1-\varphi_2)] \tag{5−9}$$

$$Q(t)=\frac{1}{2}x(t)\sin[(\omega_1-\omega_2)t+(\varphi_1-\varphi_2)] \tag{5−10}$$

可令 $\varPhi$ 表示这个输出信号的相位，即

$$\varPhi=(\omega_1-\omega_2)t+(\varphi_1-\varphi_2) \tag{5−11}$$

则本地载波与接收信号的频差就可以表示为

$$\Delta\omega=\omega_1-\omega_2=\frac{\mathrm{d}\varPhi}{\mathrm{d}t} \tag{5−12}$$

根据图 4−15 中低通滤波器 B 的输出来计算频差的过程如下：

$$\begin{aligned}\Delta\omega&=\frac{\mathrm{d}}{\mathrm{d}t}\arctan\left(\frac{Q(t)}{I(t)}\right)\\&=\frac{1}{1+\left(\dfrac{Q(t)}{I(t)}\right)^2}\cdot\frac{\mathrm{d}}{\mathrm{d}t}\frac{Q(t)}{I(t)}\\&=\frac{I(t)\dfrac{\mathrm{d}Q(t)}{\mathrm{d}t}-Q(t)\dfrac{\mathrm{d}I(t)}{\mathrm{d}t}}{I(t)^2+Q(t)^2}\end{aligned} \tag{5−13}$$

如果用式（5−13）来计算频差，在信号比较小的时候，会得到很大的误差。为了避免这个问题，解调器的 AGC 电路做了如下处理。当信号幅度比 1 大时，令信号除以平均幅度，即载波恢复电路的输入的平均幅度接近 1，或者说 $I(t)^2+Q(t)^2\approx 1$。当信号幅度小于 1 时，AGC 电路不对信号进行处理，这时，如果直接使用式（5−13）来计算 $\Delta\omega$，误差仍然会很大，而且信号幅度小的时候正是绝对相位不同的相邻符号切换的时刻，此时的频差也会出现较大的误差。因此，仍然令 $I(t)^2+Q(t)^2\approx 1$，这样计算出来的 $\Delta\omega$ 就会变得比较小，因而对频率恢复影响就会很小，从而保证了系统的稳定性。综上所述，可以用下面的式子来计算

频差：

$$\Delta\omega' = I(t)\frac{\mathrm{d}Q(t)}{\mathrm{d}t} - Q(t)\frac{\mathrm{d}I(t)}{\mathrm{d}t} \tag{5-14}$$

载波恢复部分的框图如图 5-3 所示。

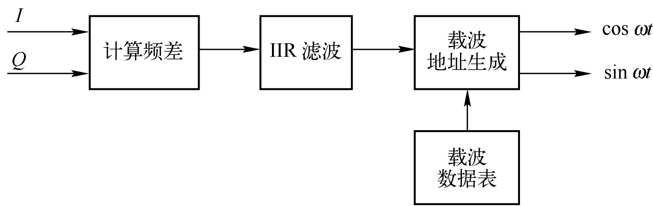


图 5-3　载波恢复框图

式（5-14）是模拟信号的计算方法，对于数字系统来说实现方法稍有不同，所采用的公式如下：

$$\Delta\omega' = I_n(Q_{n+1} - Q_{n-1}) - Q_n(I_{n+1} - I_{n-1}) \tag{5-15}$$

用 3 个相邻的采样点来实现式（5-14），用 I_n 代替 $I(t)$，用 Q_n 代替 $Q(t)$，用 $(Q_{n+1} - Q_{n-1})$ 代替 $\frac{\mathrm{d}Q(t)}{\mathrm{d}t}$，用 $(I_{n+1} - I_{n-1})$ 代替 $\frac{IQ(t)}{\mathrm{d}t}$，其中 $\mathrm{d}t$ 被忽略了。这是因为在数字系统中，$\mathrm{d}t$ 是个固定的时间值，即是个常数，而我们需要的频差也只能是个相对值，因此计算结果的一个比例系数是不重要的。而这样还带来额外的好处是计算量减少了。

IIR 滤波的具体特性可参考第 3 章相关内容。

本地载波的生成与第 4 章中的 4.2.3 节的部分是基本类似的，也是采用了查表法。唯一的区别是每个采样间隔内载波的相位增量不再是固定的值，而是在用接收频率计算的相位增量基础上用频差修正，即

$$m_NCOphzinc = \frac{2\pi \times m_RxFreq}{8000} + k \times \Delta\omega' \tag{5-16}$$

式中，m_RxFreq 是接收频率；$m_NCOphzinc$ 是相邻采样之间的载波的相位增量；$\Delta\omega'$ 是从式（5-15）计算得到并经过 IIR 滤波的频差估计；k 是比例系数。k 需要精心选择，这个参数太大了调整幅度就大，同步时间会短一些，会令本地载波在接收信号的载波附近波动过大，这个参数如果太小了，本地载波会比较接近接收信号的载波，但是同步过程会比较长。为此，这个系数在刚开始接收时采用一个较大的值（$k = 3\times10^{-6}$），令系统快速捕获，等接收了 16 个符号后，采用较小的系数（$k = 1.5\times10^{-6}$），令系统能稳定地跟踪接收信号的载波变化。

5.2　位同步

位同步也同样对解调器的性能有很大的影响。位同步的目的是实现接收方与发送方的位同步信号同频、同相。

因为数字解调器的基本工作过程都是对基带信号进行采样、判决，进而得到数字信息

的。如果接收方的位同步信号与发送方不同频，就会令接收方在原来的一个符号的时间范围内取样、判决的次数不严格地等于一次，最终导致多接收或少接收数据，引起信息的错误。这也就是通常所说的“滑码”现象。

位同步的同相也有着重要的意义。由于现代数字调制解调器普遍采用了脉冲成型技术，即意味着一个符号的时间范围内，基带信号的幅度是不相同的，因此在一个符号的范围内存在一个相对最佳的取样、判决时刻。在最佳时刻，信号的幅度较大，符号间的串扰也相对小，因此可以得到最好的误码性能。

5.2.1　位同步的方法

实现位同步的方法也和载波同步类似，可分为插入导频法和直接法两类。这两类方法有时也分别称为外同步法和自同步法。

基带信号若为随机的二进制不归零脉冲序列，那么这种信号本身不包含位同步信号。为了获得位同步信号，就应在基带信号中插入位同步导频信号，或者对该基带信号进行某种变换。

1. 插入导频法

插入导频法与载波同步时的插入导频法类似，它是在基带信号频谱的零点插入所需的导频信号，如图 5−4a 所示。若经某种相关编码的基带信号，其频谱的第一个零点在 f=1/2T 处时，插入导频信号就应在 1/2T 处，如图 5−4b 所示。

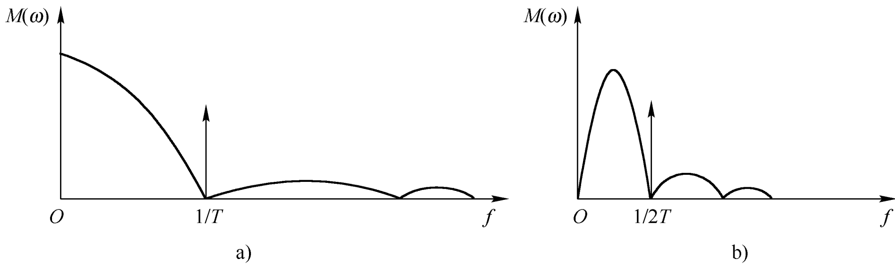


图 5−4　插入导频法频谱图

在接收端，对图 5−4a 所示的情况，经中心频率为 f=1/T 的窄带滤波器，就可从解调后的基带信号中提取出位同步所需的信号，这时，位同步脉冲的周期与插入导频的周期是一致的；对图 5−4b 所示的情况，窄带滤波器的中心频率应为 1/2T，因为这时位同步脉冲的周期为插入导频周期的 1/2，故需将插入导频倍频，才得到所需的位同步脉冲。插入导频法的另一种形式是使数字信号的包络按位同步信号的某种波形变化。

在相移键控或频移键控的通信系统中，对已调信号进行附加的幅度调制后，接收端只要进行包络检波，就可以形成位同步信号。

设相移信号的表示式为

$$s_1(t)=\cos[\omega_c t+\varphi(t)] \tag{5−17}$$

现在用某种波形的位同步信号对$s_1(t)$进行幅度调制，若这种波形为升余弦波形，则其表达式为

$$m(t)=\frac{1}{2}(1+\cos\Omega t) \tag{5-18}$$

式中，$\Omega=2\pi/T, T$ 为码元宽度。幅度调制后的信号为

$$s_2(t)=\frac{1}{2}(1+\cos\Omega t)\cos[\omega_c t+\varphi(t)] \tag{5-19}$$

接收端对$s_2(t)$进行包络检波，包络检波器的输出为 $\frac{1}{2}(1+\cos\Omega t)$，除去直流分量后，可获得位同步信号 $\frac{1}{2}\cos\Omega t$。

以上载波同步和位同步中所采用的导频插入法都是在频域内的插入。事实上，同步信号也可以在时域内插入，这时载波同步信号、位同步信号和数据信号分别被配置在不同的时间内传送。接收端用锁相环路提取出同步信号并保持它，就可以对随之而来的数据进行解调了。

2. 直接法

直接法是发送端不专门发送导频信号，而直接从数字信号中提取位同步信号的方法。这是数字通信中经常采用的一种方法。

（1）滤波法

我们知道，对于不归零的随机二进制序列，不能直接从其中滤出位同步信号。但是，若对该信号进行某种变换，例如，变成归零脉冲后，则该序列中就有 $f=1/T$ 的位同步信号分量，其大小可算出。经一个窄带滤波器，可滤出此信号分量，再将它通过一移相器调整相位后，就可以形成位同步脉冲。这种方法的原理框图如图 5–5 所示。它的特点是先形成含有位同步信息的信号，再用滤波器将其滤出。下面介绍几种具体的实现方法。

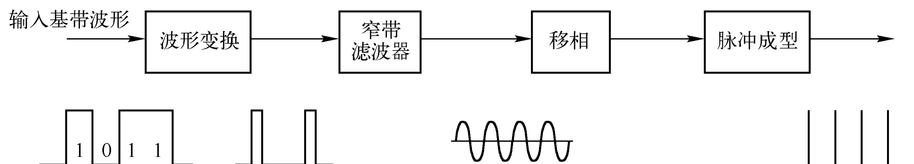


图 5–5　滤波法原理框图

图 5–5 原理图中的波形变换，在实际应用中可以是一个微分、整流电路，经微分、整流后的基带信号波形如图 5–6 所示。这里，整流输出的波形与图 5–5 中波形变换电路的输出波形有些区别，这个波形同样包含有位同步信号分量。

另一种常用的波形变换方法是对带限信号进行包络检波。在某些数字微波中继通信系统中，经常在中频上用对频带受限的二相移相信号进行包络检波的方法来提取位同步信号。频带受限的二相 PSK 信号波形如图 5–7a 所示。因频带受限，在相邻码元的相位变换点附近会产生幅度的平滑“陷落”。经包络检波后，可得图 5–7b 所示的波形。可以看出，它是一直流和图 5–7c 所示的波形相减而得到的，因此包络检波后的波形中包含有如图 5–7c 所示的波形，而这个波形中已含有位同步信号分量。因此，将它经滤波器后就可提取出位同步信号。

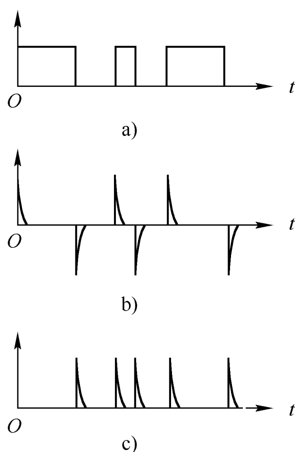


图 5-6　基带信号微分、整流波形

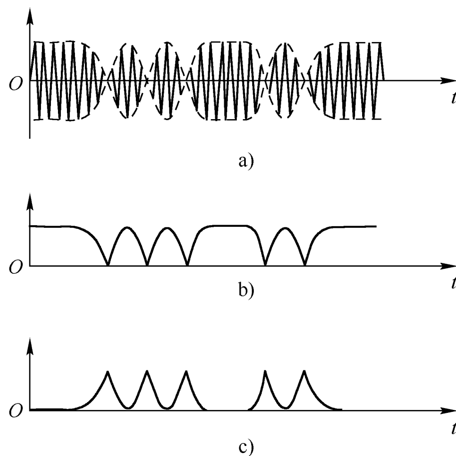


图 5-7　频带受限 2PSK 位同步信号提取

（2）锁相法

位同步锁相法的基本原理和载波同步法类似。在接收端利用鉴相器比较接收码元和本地产生的位同步信号的相位，若两者相位不一致（超前或滞后），鉴相器就产生误差信号去调整位同步信号的相位，直至获得准确的位同步信号为止。前面讨论的滤波法原理图中，窄带滤波器可以是简单的单调谐回路或晶体滤波器，也可以是锁相环路。

把采用锁相环来提取位同步信号的方法称为锁相法。下面介绍在数字通信中常采用的数字锁相法提取位同步信号的原理。

数字锁相的原理框图如图 5-8 所示，它由高稳定度振荡器（晶振）、分频器、相位比较器和控制器组成。其中，控制器包括图中的扣除门、附加门和“或门”。高稳定度振荡器产生的信号经整形电路变成周期性脉冲，然后经控制器再送入分频器，输出位同步脉冲序列。若接收码元的速率为 F（Baud），则要求位同步脉冲的重复速率也为 F（Hz）。这里，晶振的振荡频率设计在 nF（Hz），由晶振输出经整形得到重复频率为 nF（Hz）的窄脉冲（见图 5-9a），经扣除门、或门和 n 次分频器后，就可得到重复频率为 F（Hz）的位同步信号（见图 5-9c）。

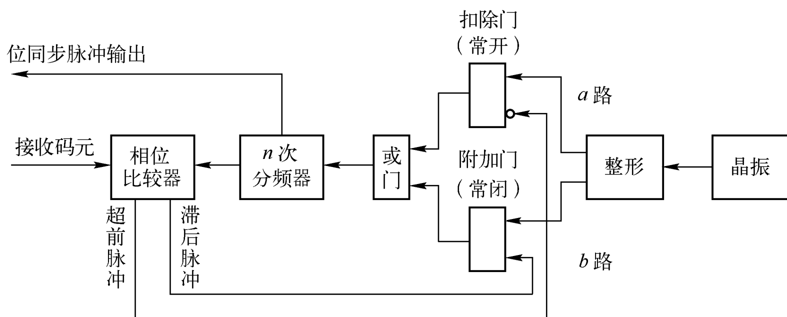


图 5-8　数字锁相的原理框图

如果接收端晶振输出经n次分频后，不能准确地和收到的码元同频同相，这时就要根据相位比较器输出的误差信号，通过控制器对分频器进行调整。调整的原理是当分频器输出的位同步脉冲超前于接收码元的相位时，相位比较器送出一超前脉冲，加到扣除门（常开）的

禁止端，扣除一个a路脉冲（见图 5-9d），这样，分频器输出脉冲的相位就推后 1/n周期（360°/n），如图 5-9e所示；若分频器输出的位同步脉冲相位滞后于接收码元的相位，如何对分频器进行调整呢？晶振的输出整形后除a路脉冲加于扣除门外，同时还有与a路相位相差180°的b路脉冲序列（见图 5-9b）加于附加门。附加门在不调整时是封闭的，对分频器的工作不起作用。当位同步脉冲相位滞后时，相位比较器送出一滞后脉冲，加于附加门，使b路输出的一个脉冲通过“或门”，插入在原a路脉冲之间（见图 5-9f），使分频器的输入端添加了一个脉冲。于是，分频器的输出相位就提前 1/n周期（见图 5-9g）。经这样的反复调整相位，就实现了位同步。

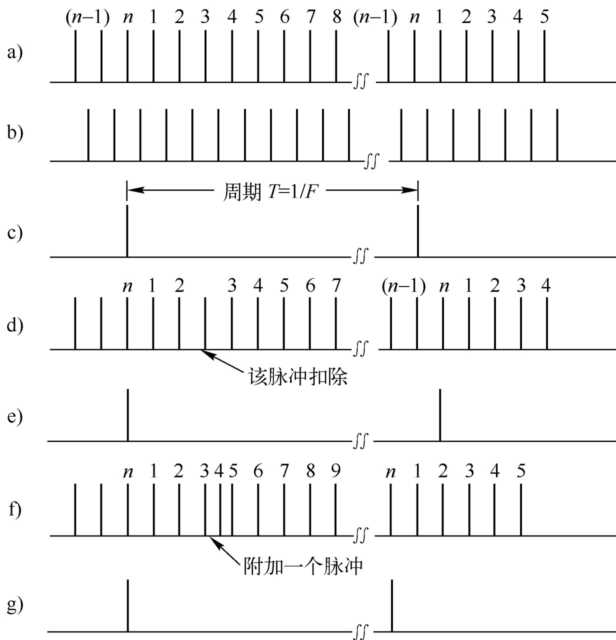


图 5-9 位同步脉冲的相位调整

（3）超前-滞后门同步器

超前-滞后符号同步算法利用的是信号波形的对称性，即经过匹配滤波器或相关器后，输出信号是对称的，如图 5-10 所示。对于图 5-10a 所示的矩形脉冲，匹配滤波器的输出在t=T 时达到最大值，如图 5-10 b 所示。只要采样值在峰值上，就一定能够保证符号同步。

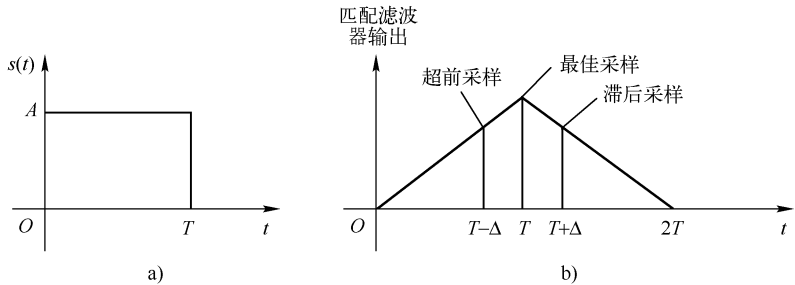


图 5-10 超前-滞后门波形示意图

a) 矩形信号脉冲 b) 匹配滤波器输出

在噪声存在的情况下，如果没有在峰值点对信号采样，而在 $t=T-\varDelta$ 时超前采样，在 $t=T+\varDelta$ 时滞后采样。那么由于自相关函数对于最佳采样时刻 $t=T$ 是偶函数，超前采样值的绝对值和滞后采样值的绝对值就相等。在这种条件下，适当的采样时刻应该是在 $t=T-\varDelta$ 和 $T+\varDelta$ 的中间。这一条件构成了超前–滞后门同步器的基础。

超前–滞后门同步器的结构如图 5–11 所示。

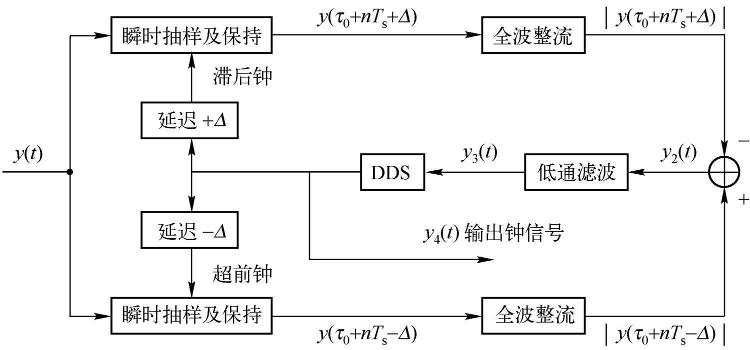


图 5–11　超前–滞后门同步器的结构示意图

下面对其进行简单的说明。

根据信号的特点，信号的波形是对称于最佳采样时刻的。本方法是利用信号脉冲波形对称性的特点来进行符号同步的。在图 5–11 中，用 $y(t)$ 表示接收滤波器的输出信号波形，假设在眼图张开到最大时进行采样，即在最佳时刻进行采样，得到的采样值为 $y(\tau_0+nT_s)$，τ_0 是最佳定位相位。

设 $\varDelta$ 是偏离于最佳采样时刻的偏离值，如图 5–11 所示，在偏离值为 $\varDelta$ 的两个采样时刻是相等的，一个为超前采样，用 $y(\tau_0+nT_s-\varDelta)$ 表示；另一个为滞后采样，用 $y(\tau_0+nT_s+\varDelta)$ 表示，即

$$|y(\tau_0+nT_s-\varDelta)|\approx|y(\tau_0+nT_s+\varDelta)|$$

但是在未同步时的采样相位 $\tau\neq\tau_0$，此时的超前采样为 $y(\tau_0+nT_s-\varDelta)$，滞后采样为 $y(\tau+nT_s+\varDelta)$，分别将它们全波整流，得到 $|y(\tau_0+nT_s-\varDelta)|$ 及 $|y(\tau_0+nT_s+\varDelta)|$。再将两者相减，得到

$$y_2(t)=|y(\tau_0+nT_s-\varDelta)|-|y(\tau_0+nT_s+\varDelta)|$$

再将 $y_2(t)$ 经过低通滤波，相当于对 $y_2(t)$ 进行时间平均，得到输出 $y_3(t)$。

将 $y_3(t)$ 送给DDS，控制DDS的频率。若DDS产生的时钟是最佳定位相位，即 $\tau=\tau_0$，则 $y_3(t)$ 为 0；若 τ 超前于 τ_0，则 $y_3(t)$ 是负值；若 τ 滞后于 τ_0，则 $y_3(t)$ 是正值。正的控制电压将增加DDS的频率，负的电压将减小DDS的频率。在符号同步时，DDS将输出时钟信号，同步于接收信号的符号速率。注意，该电路要避免输入数据全“1”或“0”的情况。

基于超前–滞后门的符号同步算法在无线通信信号处理中获得了广泛应用，下面我们就以此为例，用 MATLAB 进行实现，以期加深读者的理解。

5.2.2 位同步的 MATLAB 实现

【例 5-2】 用MATLAB实现基于超前-滞后门的符号同步算法。

```
clear
Fs=500;                                         %采样频率
nsamp=16;                                       %过采样率
delay=3;                                        %平方根升余弦的群时延
DI=randint(1,1000)+1i*randint(1,1000);          %输入的数据
num=length(DI);
DO=PRCsend(DI,Fs,nsamp,delay);                  %调用子程序，得到发送滤波后信号
SNR=22;                                         %信噪比
snr=1/(10^(SNR/10));
noise_var=2/snr;
noise=sqrt(1/2*noise_var)*randn(1,length(DO))+1i*randn(1,length(DO));
datachannel=DO'+noise;                          %加噪声

DR=PRCrece(datachannel,Fs,nsamp,delay,1);       %调用子程序，得到接收滤波后信号
opt_point=1+2*delay*nsamp;
DE=[DI,DI,DI];                                  %接收进行相关的数据

[b,a]=cheby2(3,20,100/19200);                   %设计 3 阶，带外衰减 20dB，低通带宽为 100Hz

for ii=1:16
    DS=DR(opt_point-16+ii:16:length(DR)-2*delay*nsamp-15);
    temp = real(DS').*real(DE(num:2*num-1))+imag(DS').*imag(DE(num:2*num-1));
    temp=filter(b,a,temp);                      %经过低通滤波器
    val_co1=sum(temp);                          %超前门支路相关
    temp=real(DS').*real(DE(num+2:2*num+1))+imag(DS').*imag(DE(num+2:2*num+1));
    temp=filter(b,a,temp);                      %经过低通滤波器
    val_co2=sum(temp);                          %超前门支路相关
    val_co(ii)=(val_co2-val_co1)/(2*num);       %计算两个支路差值，并归一化
end

for ii=1:15
    DS=DR(opt_point+ii:16:length(DR)-2*delay*nsamp+1);
    temp = real(DS').*real(DE(num:2*num-1))+imag(DS').*imag(DE(num:2*num-1));
    temp=filter(b,a,temp);                      %经过低通滤波器
    val_co1=sum(temp);                          %超前门支路相关
    temp=real(DS').*real(DE(num+2:2*num+1))+imag(DS').*imag(DE(num+2:2*num+1));
    temp=filter(b,a,temp);                      %经过低通滤波器
    val_co2=sum(temp);                          %超前门支路相关
    val_co(16+ii)=(val_co2-val_co1)/(2*num);    %计算两个支路差值，并归一化
end

ii=-15:15;
figure;
```

```
stem(ii,val_co);
xlabel('采样点偏移');
ylable('归一化相关值');
```

发送滤波子程序：

```
function dataout= PRCsend(datain,Fs,nsamp,delay)

num = rcosine(Fs,Fs*nsamp,'fir/sqrt',0.22,delay);
dataoutr=rcosflt(real(datain),Fs,Fs*nsamp,'filter',num);
dataouti=rcosflt(imag(datain),Fs,Fs*nsamp,'filter',num);
dataout=(dataoutr+1i*dataouti);
```

接收滤波子程序：

```
function dataout= PRCrece(datain,Fs,nsamp,delay,carriers)

num = rcosine(Fs,Fs*nsamp,'fir/sqrt',0.22,delay);
dataoutr=rcosflt(real(datain),Fs,Fs*nsamp,'Fs/filter',num);
dataouti=rcosflt(imag(datain),Fs,Fs*nsamp,'Fs/filter',num);
dataout=(dataoutr+1i*dataouti);
```

运行上述程序，就可以得到如图 5-12 所示的 S 曲线，图中归一化相关值为零的点即为超前-滞后门的收敛点，在这个时刻进行接收信号采样时，它的超前采样值和滞后采样值相等，即为最佳采样时刻。

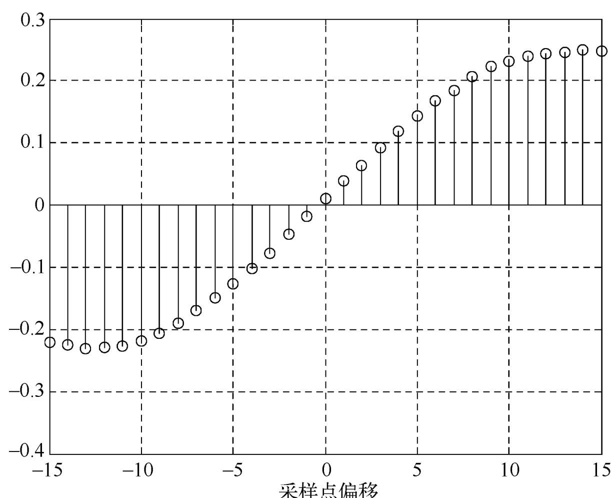


图 5-12　S 曲线图

5.2.3　位同步的 dsPIC 实现

针对第 4 章设计的调制解调器，位同步可以把同频的功能省略。这主要是因为该调制解调器工作在半双工模式，其连续工作在收状态或发状态的时间不会很长，另一方面，现在最

普通的晶体振荡器都能达到 10^{-5}以上的精度级别，而且该调制解调器的符号速率固定在 31.25 符号/s，因此，同频的功能在这里是可以被省略的。在一次短促的接收结束之前，这样微小的定时误差是不足以引起“滑码”的。

位同步的同相功能是不能忽略的，因为调制器发送的基带波形在绝对相位发生变化时是余弦加权的，也就是说在一个符号的时间范围内，信号的幅度是不同的，存在一个最大值。为了获得良好的误码性能，必须找到那个最大值的位置。

在使用 dsPIC 实现位同步的同相时，采用了一种简化的方案。因为一个符号时间的范围对应 16 个连续的采样点。所以，在接收端，每接收 16 个采样点算做一组，在这一组内找信号能量最大的那个采样点，就是位同步脉冲所在点。这样也就避开了繁琐的锁相环的实现，令系统十分简洁。具体实现的框图如图 5−13 所示。

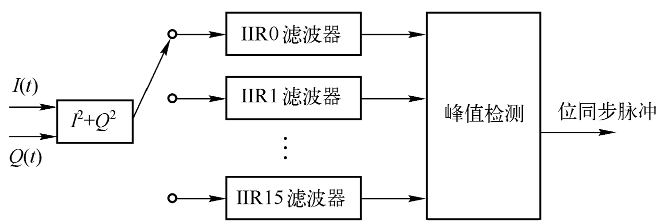


图 5−13　位同步的 dsPIC 实现框图

为了计算每个采样点的信号能量大小，采用了在复平面上计算信号幅度的平方的方法。把这些信号的能量值每 16 个分为一组，每组中第 0 个被送到 IIR0 滤波器，第 1 个被送到 IIR1 滤波器，以此类推，分别被送到 16 个 IIR 滤波器。在滤波器的输出端进行峰值检测，即在这 16 个值中的最大点就被定为解调器中的判决点。

图 5−13 中的 IIR 滤波器的作用是对信号能量进行平滑滤波，以避免由于噪声的影响，而引起的判决点的随机抖动。IIR 滤波器的具体特性参见第 3 章相关内容。

5.3　帧同步

数字通信时，一般总是以一定数目的码元组成一个个的“字”或句，即组成一个个的“群”进行传输，因此群同步信号的频率很容易由位同步信号经分频得出，但是，每群的开头和结尾时刻却无法用分频器的输出决定。群同步的任务就是要给出这个“开头”和“结尾”的时刻。群同步有时又称为帧同步。

5.3.1　群同步的方法

为了实现群同步通常有两类方法：一类是在数字信息流中加入一些特殊码组作为每群的头尾标记，接收端根据这些特殊码组的位置就可以实现群同步；另一类方法不需要外加的特殊码组，它类似于载波同步和位同步中的直接法，利用各数据码组彼此不同的特性来实现自同步。本节将主要讨论用插入特殊码组实现群同步的方法。

插入特殊码组实现群同步的方法有两种，即连贯式插入法和间隔式插入法。在介绍这两种方法之前，先介绍一种首先在电传打字机中广泛使用的起止式群同步法。

1. 起止式群同步法

电传打字机的一个字由 7.5 个码元组成，如图 5−14 所示。每个字开头，先发一个码元的起脉冲（负值），中间 5 个码元是消息，字的末尾是 1.5 码元宽度的止脉冲（正值），收端根据正电平第一次转到负电平这一特殊规律，确定一个字的起始位，因而就实现了群同步。由于这种方式的止脉冲宽度与码元宽度不一致，就会给同步数字传输带来不便。另外，在这种同步方式中，7.5 个码元中只有 5 个码元用于传递消息，因此效率较低。

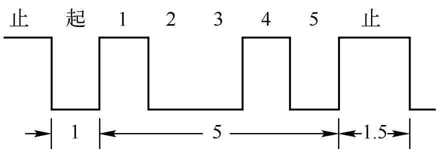


图 5−14　起止式同步的信号波形

2. 连贯式插入法

连贯式插入法就是在每群的开头集中插入群同步码组的方法。作群同步码组用的特殊码组首先应该是具有尖锐单峰特性的局部自相关函数。由于这个特殊码组$\{x_1,x_2,x_3,\cdots x_n\}$是一个非周期序列或有限序列，在求它的自相关函数时，除了在时延$j=0$ 时，序列中的全部元素都参加相关运算外，在$j\neq0$ 的情况下，序列中只有部分元素参加相关运算，其表达式为

$$R(j)=\sum_{i=1}^{n-j}x_i x_{i+j} \tag{5−20}$$

通常把这种非周期序列的自相关函数称为局部自相关函数。对同步码组的另一个要求是识别器应该尽可能简单。目前，一种常用的群同步码组是巴克码。

巴克码是一种非周期序列。一个n位的巴克码组为$\{x_1,x_2,x_3,\cdots,x_n\}$，其中，$x_i$取值为＋1 或−1，它的局部自相关函数为

$$R(j)=\sum_{i=1}^{n-j}x_i x_{i+j}=\begin{cases} n, & j=0 \\ 0\text{或}\pm1, & 0<j<n \\ 0, & j \ldots n \end{cases} \tag{5−21}$$

目前所找到的所有巴克码组列于表 5−1 中。

表 5−1　所有巴克码组

n	巴克码组
2	++
3	++−
4	+++−;++−+
5	+++−+
7	+++−−+−
11	+++−−−+−−+−
13	+++++−−++−+−+

以 7 位巴克码组{+++−−+−}为例，求出它的自相关函数如下：

当 $j=0$ 时，$R(j)=\sum_{i=1}^{7}x_i^2=1+1+1+1+1+1+1=7$

当 $j=1$ 时， $R(j)=\sum_{i=1}^{6}x_i x_{i+1}=1+1-1+1-1+1-1=0$

按式（5−20）可求出 j=2,3,4,5,6,7 时的 $R(j)$值分别为−1，0，−1，0，−1，0；再求出 j 为负值时的自相关函数值，两者一起画在图 5−15 中。由图可见，其自相关函数在 j=0 时出现尖锐的单峰。

巴克码识别器是比较容易实现的，这里以 7 位巴克码为例，使用 7 级移位寄存器、相加器和判决器来实现，如图 5−16 所示。

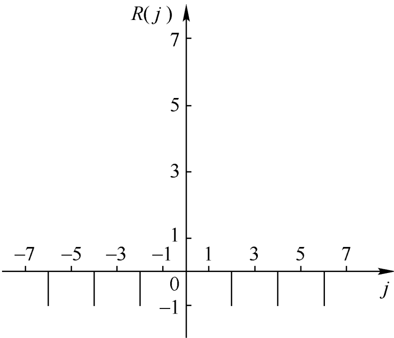


图 5−15　7 位巴克码的自相关函数

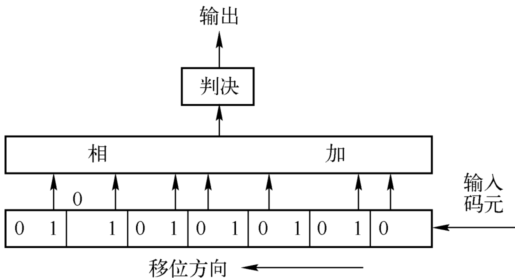


图 5−16　7 位巴克码识别器

当输入数据的“1”存入移位寄存器时，“1”端的输出电平为+1，而“0”端的输出电平为−1；反之，存入数据“0”时，“0”端的输出电平为+1，“1”端的电平为−1。各移位寄存器输出端的接法和巴克码的规律一致，这样识别器实际上就是对输入的巴克码进行相关运算。当 7 位巴克码在图 5−17a中的t_1时刻正好全部进入了 7 级移位寄存器时，7 个移位寄存器的输出端都输出+1，相加后最大输出为+7；若判决器的判决门限电平为+6，那么就在 7 位巴克码的最后一位“0”进入识别器时，识别器输出一同步脉冲表示一群的开头，如图 5−17 所示。

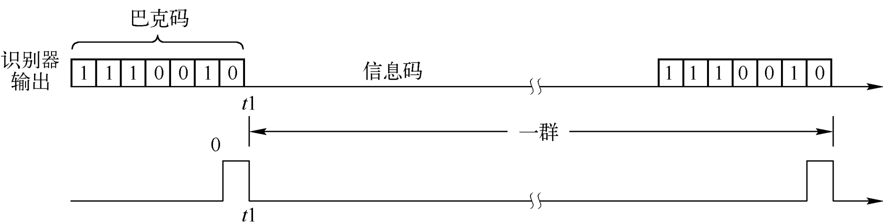


图 5−17　识别器的输出波形

帧同步或称群同步是在调制解调器输出的二进制比特流基础上的同步。这与调制解调器的工作场合、系统的要求有很大的关系。其关键是设计出一套高效、易于实现的方案。帧同步的实现需要在调制前对原始信息进行编码并输出二进制比特流；在解调后再对二进制比特流进行解码并输出原始信息。因此本节分为帧同步的编码与解码两部分来介绍。

5.3.2　帧同步编码的 MATLAB 实现

【例 5−3】 使用MATLAB实现基于 7 位巴克码的集中插入式帧同步算法。

由于可能出现信息序列和巴克码相同的情况，因此为了同步的准确性，系统在捕捉到帧头后应该进入维持状态，即仍然需要逐帧比较下去，直到出现帧头的位置始终周期出现。

在本例中为了简单起见，只利用了 3 帧数据。假设判定门限为 6，此数值可针对不同系统的要求进行调整。进行多次仿真可以得到正确的帧同步。相应的程序如下：

```
clear
len=100;                            %每帧数据长度
n=3;                                %进行仿真的数据帧数
c=[1 1 1 -1 -1 1 -1];               %7 位巴克码
for i=1:n
    s(i,:)=randsrc(1,len);          %产生随机的数据
end
r=[c s(1,:) c s(2,:) c s(3,:)];     %将巴克码插入到数据中
thr=6;                              %判决门限
out=zeros(1,length(r));
num=[];                             %存储可能的帧头的位置
for i=7:n*len +n*7
    out(i)=sum(r(i-6:i).*c);        %每次 7 个数据进行滑动相关
    if(out(i)>thr)                  %判断相关值是否大于门限
        num=[num i];                %将大于门限的位置存储起来
    end
end
num
```

运行程序，可以得到帧头位置为 num=[7 55 114 221]，即帧头位置可能在发送数据的第 7、55、114 和 221 位。反复进行多次仿真，可以发现第 7、114 和 221 位反复出现，故可以确定帧头在第 4、114 和 221 位。

5.3.3 帧同步编码的 dsPIC 实现

本书所设计的调制解调器是用来发送键盘产生的 ASCII 码的，这些 ASCII 码可由一些常用的代码构成，当然也可以用来表示人类的自然语言。因此，帧同步必须设计一种表示方法，即用一组二进制比特信息表示 256 个 ASCII 码，并且要能从连续的二进制码流中方便地分离出这些二进制比特信息组，并进一步得到 ASCII 码。

为此设计出以下的规则：

- 当不发送 ASCII 码时，调制解调器发送连“0”码。
- 任何要发送的 ASCII 码用一组二进制比特信息表示，一组二进制比特信息的开始和结尾均必须是“1”。
- 任何两个 ASCII 码之间至少有两个连续的“0”码。
- 一组二进制比特信息中间不能出现两个以上（包括两个）的连“0”码。
- 对一组二进制比特信息的长短不做特别的要求。

这样，在接收端平时收到的是“0”，当收到一个“1”时，就开始接收一个 ASCII 码。当接收到两个连续的“0”后，接收 ASCII 码结束。然后就能根据接收到的一组二进制比特信息来查找到相应的 ASCII 码了。

为了尽量提高效率，对 ASCII 码进行的编码一般采用可变长度编码，对于在会话中出现概率大的 ASCII 码就用尽量短的一组二进制信息，比如字母“e”就用“11”编码，空格就用“1”编码；而出现概率小的 ASCII 码就用比较长的一组二进制信息来编码，比如“@”就用“1010111101”编码。这样会令系统的效率大为提高。

例如要发送的信息序列是“abc”，则编码如下：

···0010110010111110010111100···

a b c

目前采用的编码方案如表 5–2 所示。

表 5–2 帧同步编码规则

输入的 ASCII 码	输出的二进制信息组	输入的 ASCII 码	输出的二进制信息组	输入的 ASCII 码	输出的二进制信息组	输入的 ASCII 码	输出的二进制信息组
NULL	1010101011	'@'	1010111101	128	1110111101	192	11011101111
SOH	1011011011	'A'	1111101	129	1110111111	193	11011110101
STX	1011101101	'B'	11101011	130	1111010101	194	11011110111
ETX	1101110111	'C'	10101101	131	1111010111	195	11011111011
EOT	1011101011	'D'	10110101	132	1111011011	196	11011111101
ENQ	1101011111	'E'	1110111	133	1111011101	197	11011111111
ACK	1011101111	'F'	11011011	134	1111011111	198	11101010101
BEL	1011111101	'G'	11111101	135	1111101011	199	11101010111
BS	1011111111	'H'	101010101	136	1111101101	200	11101011011
HT	11101111	'I'	1111111	137	1111101111	201	11101011101
LF	11101	'J'	111111101	138	1111110101	202	11101011111
VT	1101101111	'K'	101111101	139	1111110111	203	11101101011
FF	1011011101	'L'	11010111	140	1111111011	204	11101101101
CR	11111	'M'	10111011	141	1111111101	205	11101101111
SO	1101110101	'N'	11011101	142	1111111111	206	11101110101
SI	1110101011	'O'	10101011	143	10101010101	207	11101110111
DLE	1011110111	'P'	11010101	144	10101010111	208	11101111011
DC1	1011110101	'Q'	111011101	145	10101011011	209	11101111101
DC2	1110101101	'R'	10101111	146	10101011101	210	11101111111
DC3	1110101111	'S'	1101111	147	10101011111	211	11110101011
DC4	1101011011	'T'	1101101	148	10101101011	212	11110101101
NAK	1101101011	'U'	101010111	149	10101101101	213	11110101111
SYN	1101101101	'V'	110110101	150	10101101111	214	11110110101
ETB	1101010111	'W'	101011101	151	10101110101	215	11110110111
CAN	1101111011	'X'	101110101	152	10101110111	216	11110111011
EM	1101111101	'Y'	101111011	153	10101111011	217	11110111101
SUB	1110110111	'Z'	1010101101	154	10101111101	218	11110111111
ESC	1101010101	'['	111110111	155	10101111111	219	11111010101

（续）

输入的 ASCII 码	输出的二进制信息组	输入的 ASCII 码	输出的二进制信息组	输入的 ASCII 码	输出的二进制信息组	输入的 ASCII 码	输出的二进制信息组
FS	1101011101	'\'	111101111	156	10110101011	220	11111010111
GS	1110111011	']'	111111011	157	10110101101	221	11111011011
RS	1011111011	'^'	1010111111	158	10110101111	222	11111011101
US	1101111111	'_'	101101101	159	10110110101	223	11111011111
SPACE	1	'`'	1011011111	160	10110110111	224	11111101011
'!'	111111111	'a'	1011	161	10110111011	225	11111101101
' " '	101011111	'b'	1011111	162	10110111101	226	11111101111
'#'	111110101	'c'	101111	163	10110111111	227	11111110101
'$'	111011011	'd'	101101	164	10111010101	228	11111110111
'%'	1011010101	'e'	11	165	10111010111	229	11111111011
'&'	1010111011	'f'	111101	166	10111011011	230	11111111101
'''	101111111	'g'	1011011	167	10111011101	231	11111111111
'('	11111011	'h'	101011	168	10111011111	232	101010101011
')'	11110111	'i'	1101	169	10111101011	233	101010101101
'*'	101101111	'j'	111101011	170	10111101101	234	101010101111
'+'	111011111	'k'	10111111	171	10111101111	235	101010110101
','	1110101	'l'	11011	172	10111110101	236	101010110111
'–'	110101	'm'	111011	173	10111110111	237	101010111011
'.'	1010111	'n'	1111	174	10111111011	238	101010111101
'/'	110101111	'o'	111	175	10111111101	239	101010111111
'0'	10110111	'p'	111111	176	10111111111	240	101011010101
'1'	10111101	'q'	110111111	177	11010101011	241	101011010111
'2'	11101101	'r'	10101	178	11010101101	242	101011011011
'3'	11111111	's'	10111	179	11010101111	243	101011011101
'4'	101110111	't'	101	180	11010110101	244	101011011111
'5'	101011011	'u'	110111	181	11010110111	245	101011101011
'6'	101101011	'v'	1111011	182	11010111011	246	101011101101
'7'	110101101	'w'	1101011	183	11010111101	247	101011101111
'8'	110101011	'x'	11011111	184	11010111111	248	101011110101
'9'	110110111	'y'	1011101	185	11011010101	249	101011110111
':'	11110101	'z'	111010101	186	11011010111	250	101011111011
';'	110111101	'{'	1010110111	187	11011011011	251	101011111101
'<'	111101101	'|'	110111011	188	11011011101	252	101011111111
'='	1010101	'}'	1010110101	189	11011011111	253	101101010101
'>'	111010111	'～'	1011010111	190	11011101011	254	101101010111
'?'	1010101111	DEL	1110110101	191	11011101101	255	101101011011

设计如下的数组，数组中每一项是一个短整型数（占用 16 比特），代表一个 ASCII 编码。编码从最高位算起，当出现连续两个“0”比特时编码结束。

```
const unsigned int VARICODE_TABLE[256] = {
        0xAAC0,    /* ASCII =    0  1010101011          */
        0xB6C0,    /* ASCII =    1  1011011011          */
        ……
        0xB000,    /* ASCII = 'a'   1011                */
        0xBE00,    /* ASCII = 'b'   1011111             */
        0xBC00,    /* ASCII = 'c'   101111              */
        ……
};
```

当发送某个 ASCII 字符时，就从 VARICODE_TABLE[256]中取出相应的短整型数，每个符号周期发送这个短整型数的最高位，并把这个短整型数左移一位，当这个短整型数的值变成零时，该 ASCII 字符也就发送完毕了。当然，在发送一个 ASCII 字符之前和之后都要发送一个“0”码。发送过程如图 5-18 所示。

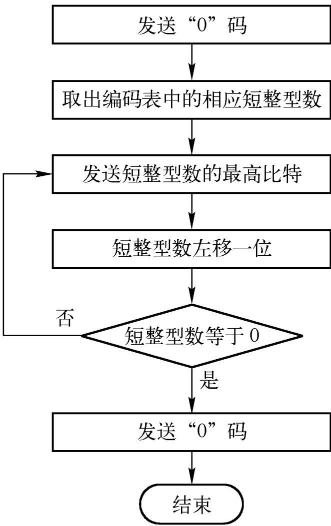


图 5-18　发送帧同步编码的过程

5.3.4　帧同步解码的 dsPIC 实现

帧同步解码的过程实际上是从接收到的二进制码流中恢复出 ASCII 字符的过程。首先根据前面的编码规则，可以很容易地从码流中找到单个的码字。在接收过程中，开始肯定接收到的都是“0”码，一旦在码流中出现了“1”码，则说明一个新的码字开始了。当码字中出现两个连续的“0”码时，该码字接收结束。此时，再根据表 5-1 的编码规则进行译码，这样就得到了发送方发出的 ASCII 码。

帧同步的解码过程如图 5-19 所示。

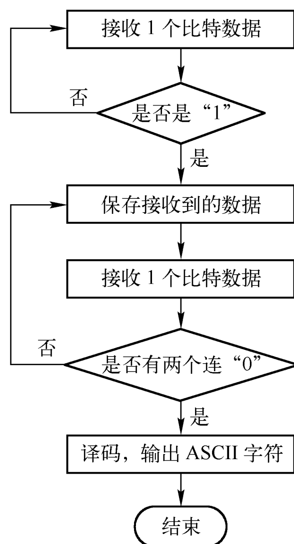


图 5−19　接收帧同步解码过程

上述解码过程中，把单个码字译成 ASCII 码的基本方法就是把接收到的码字与 VARICODE_TABLE[256]中的数进行逐个比对，直到找到对应的 ASCII 码为止。这样的算法虽然简单，但是比较耗时。

为了提高译码速度，可以采用查表法。观察表 5−1，可以发现表内最长的码字为 12 比特，这样如果建立一个长度为 4096 比特的译码数组，就可以实现查表法译码了。根据最初的编码规则，可以知道每个码字的最后一个比特都是 1，因此查表时只使用 11 个比特就够了，因此译码数组的长度可以缩短为 2048 比特。一种译码表可以表示如下：

```
m_VaricodeDecTbl[2048]={
0x20,// SPACE   1
0x65,//e      11
0x74,//t      101
0x6f,//o      111
0x20,//非法码字  1001
0x61,//a      1011
0x74,//t      1101
……
};
```

第 6 章　信道编译码器的设计与实现

差错控制编码，又称为信道编码、纠错码、抗干扰编码或可靠性编码，它是提高数字信号传输可靠性的有效方法之一。它产生于 20 世纪 50 年代，发展于 60 年代，70 年代趋于成熟。本章主要分析差错控制编码的基本方法及纠错编码的基本原理，常用检错码、线性分组码、卷积码、格状编码调制及 *m* 序列的构造原理及其应用。学习编码，一方面必须从它的构造理论来进行理解和把握，另一方面可以利用计算机对差错控制编码进行仿真。在本章中，我们将对信道编译码理论进行简要的回顾，然后对利用 MATLAB 和 dsPIC 进行编译码进行较为详细的说明。

6.1　概述

由于数字信号在传输过程中受到干扰的影响，使信号码元波形变坏，故传输到接收端后可能发生错误判决。由信道中乘性干扰引起的码间干扰，通常可以采用均衡的办法纠正，而加性干扰的影响则要通过其他途径解决。通常，在设计数字通信系统时，首先应从合理地选择调制解调方式以及发送功率等方面考虑。若采取上述措施仍难以满足要求，则就要考虑采用本章所述的差错控制措施了。从差错控制角度看，按加性干扰引起的错码分布规律的不同，信道可以分为三类，即随机信道、突发信道和混合信道。在随机信道中，错码的出现是随机的，且错码之间是统计独立的。例如，由正态分布白噪声引起的错码就具有这种性质。因此，当信道中加性干扰主要是这种噪声时，就称这种信道为随机信道。在突发信道中，错码是成串集中出现的，也就是说，在一些短促的时间区间内会出现大量错码，而在这些短促的时间区间之间却又存在较长的无错码区间。这种成串出现的错码称为突发错码。产生突发错码的主要原因之一是脉冲干扰，而信道中的衰落现象也是产生突发错码的另一主要原因。当信道中加性干扰主要是这种干扰时，便称这种信道为突发信道。把既存在随机错码又存在突发错码，且哪一种都不能忽略不计的信道，称为混合信道。对于不同类型的信道，应采用不同的差错控制技术。常用的差错控制方法有以下几种：

（1）检错重发法（ARQ）

接收端在收到的信码中检测出错码时，即设法通知发送端重发，直到正确收到为止。所谓检测出错码，是指在若干接收码元中知道有一个或多个是错的，但不一定知道该错码的准确位置。采用这种差错控制方法需要具备双向信道。

（2）前向纠错法（FEC）

接收端不仅能在收到的信码中发现有错码，还能够纠正错码。对于二进制系统，如果能够确定错码的位置，就能够纠正它。这种方法不需要反向信道（传递重发指令），也不存在由于反复重发而延误时间，实时性好。但是纠错设备要比检错设备复杂。

（3）反馈校验法（IF）

接收端将收到的信码原封不动地转发回发送端，并与原发送信码相比较。如果发现错误，则发送端再进行重发。这种方法原理和设备都较简单，但需要有双向信道。因为每一信码都相当于至少传送了两次，所以传输效率较低。

上述三种差错控制方法可以结合使用，例如，检错和纠错结合使用。当出现少量错码并在接收端能够纠正时，就用前向纠错法纠正；当错码较多而超过纠正能力但尚能检测时，就用检错重发法。此外，在某些特定场合，可采用检错删除，即接收端将其中存在错误的部分码元删除，不送给输出端。此法适用于信息内容有大量冗余度或多次重复发送的场合。在上述三种方法中，前两种方法的共同点都是在接收端识别有无错码。那么，接收端根据什么来识别呢？由于信息码元序列是一种随机序列，接收端是无法预知的（如果预先知道，就没有必要发送了），也无法识别其中有无错码。为了解决这个问题，可以由发送端的信道编码器在信息码元序列中增加一些监督码元。这些监督码和信码之间有一定的关系，使接收端可以利用这种关系由信道译码器来发现或纠正可能存在的错码。

在信息码元序列中加入监督码元就称为差错控制编码，有时也称为纠错编码。不同的编码方法，有不同的检错或纠错能力，有的编码只能检错，不能纠错。一般说来，付出的代价越大，检（纠）错的能力就越强。这里所说的代价，就是指增加的监督码元的多少，它通常用冗余度来衡量。例如，若编码序列中，平均每两个信息码元就有一个监督码元，则这种编码的冗余度为 1/3。也可以说这种编码的编码效率为 2/3。可见，差错控制编码原则上是以降低信息传输速率为代价来换取传输可靠性的提高。本章的主要内容就是讨论各种常见的编码和解码方法。

为了使读者对于具有差错控制能力的传输系统的组成有个概念，在讨论纠错编码原理之前，先简要介绍一种检错重发系统（自动要求重发系统）的组成。

自动要求重发系统通常简称为 ARQ 系统，其组成原理框图如图 6−1 所示。这种系统中应有双向信道。在发送端，输入的信息码元在编码器中被分组编码（加入监督码元）后，除立即发送外，尚暂存于缓冲存储器中。若接收端解码器检出错码，则由解码器控制产生一重发指令，经反向信道送至原发送端。这时，由发送端重发控制器控制缓冲存储器重发一次。接收端仅当解码器认为接收信息码元正确时，才将信码送给收信者，否则在输出缓冲存储器中删除掉。当接收端解码器未发现错码时，经反向信道发出不需重发指令。发送端收到此指令后，即继续发送后一码组，发送端的缓冲存储器中的内容也随之更新。

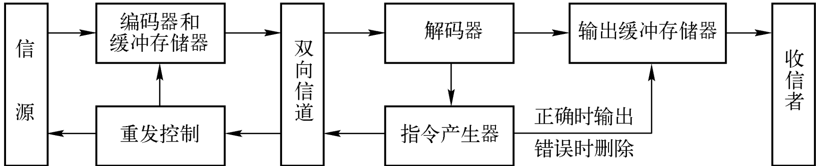


图 6−1　ARQ 系统组成原理框图

ARQ 方式的主要优点是：

1）只需要少量的多余码元（一般为总码元的 5%~20%）就能获得极低的输出误码率。

2）要求使用的检错码基本上与信道的差错统计特性无关，也就是说，对各种信道的不

同差错特性，有一定自适应能力。

3）其检错译码器与前向纠错法中的纠错译码器相比，成本和复杂性均低得多。

这种方法的主要缺点是：

1）由于需要双向信道，故不能用于单向传输系统，也难以用于广播（一发多收）系统，并且实现重发控制比较复杂。

2）当信道干扰增大时，整个系统可能处在重发循环中，因而通信效率降低至不能通信。

3）不太适于要求严格实时传输的系统。

6.2　线性分组码原理及实现

6.2.1　线性分组码的基本原理

我们把建立在代数学基础上的编码称为代数码。在代数码中，常见的是线性码。线性码中信息位和监督位是由一些线性代数方程联系着的，或者说，线性码是按一组线性方程构成的。本节将以汉明（Hamming）码为例说明线性分组码的一般原理。

汉明码是一种能够纠正一位错码且编码效率较高的线性分组码。下面介绍汉明码的构造原理。

一般说来，若码长为 n，信息位数为 k，则监督位数 $r=n-k$。如果希望用 r 个监督位构造出 r 个监督关系式来指示一位错码的 n 种可能位置，则要求

$$2^r - 1 \ldots n \quad 或者\ 2^r \ldots k + r + 1 \tag{6-1}$$

下面通过一个例子来说明如何具体构造这些监督关系式。

设分组码（n,k）中k=4。为了纠正一位错码。由式（6−1）可知，要求监督位数$r \ldots 3$。若取r=3，则n=k+r=7。用a_6，a_5,⋯，a_0表示这 7 个码元，用S_1，S_2，S_3表示三个监督关系式中的校正子，则S_1、S_2、S_3的值与错码位置的对应关系可以规定为表 6−1 所列形式（当然，也可以规定成另一种对应关系，这不影响讨论的一般性）。

表 6−1　S_1、S_2、S_3的值与错码位置的对应关系

$S_1S_2S_3$	错码位置	$S_1S_2S_3$	错码位置
001	a_0	101	a_4
010	a_1	110	a_5
100	a_2	111	a_6
011	a_3	000	无错

由表中规定可见，仅当一错码位置在a_2、a_4、a_5或a_6时，校正子S_1为 1；否则S_1为 0。这就意味着a_2、a_4、a_5和a_6四个码元构成偶数监督关系，即

$$S_1 = a_6 \oplus a_5 \oplus a_4 \oplus a_2 \tag{6-2}$$

同理，a_1、a_2、a_5和a_6构成偶数监督关系，即

$$S_2 = a_6 \oplus a_5 \oplus a_3 \oplus a_1 \tag{6-3}$$

以及a_0、a_3、a_4和a_6构成偶数监督关系，即

$$S_3 = a_6 \oplus a_4 \oplus a_3 \oplus a_0 \tag{6-4}$$

在发送端编码时，信息位a_6、a_5、a_4和a_3的值取决于输入信号，因此它们是随机的。监督位a_2、a_1和a_0应根据信息位的取值按监督关系来确定，即监督位应使以上三式中S_1、S_2和S_3的值为零（表示编成的码组中应无错码）

$$\begin{cases} a_6 \oplus a_5 \oplus a_4 \oplus a_2 = 0 \\ a_6 \oplus a_5 \oplus a_3 \oplus a_1 = 0 \\ a_6 \oplus a_4 \oplus a_3 \oplus a_0 = 0 \end{cases} \tag{6–5}$$

由上式经移项运算，解出监督位为

$$\begin{cases} a_2 = a_6 \oplus a_5 \oplus a_4 \\ a_1 = a_6 \oplus a_5 \oplus a_3 \\ a_0 = a_6 \oplus a_4 \oplus a_3 \end{cases} \tag{6–6}$$

给定信息位后，可直接按上式算出监督位，其结果如表 6–2 所列。

表 6–2　信息位与监督位对应表

信　息　位	监　督　位	信　息　位	监　督　位
$a_6a_5a_4a_3$	$a_2a_1a_0$	$a_6a_5a_4a_3$	$a_2a_1a_0$
0000	000	1000	111
0001	011	1001	100
0010	101	1010	010
0011	110	1010	001
0100	110	1100	001
0101	101	1101	010
0110	011	1110	100
0111	000	1111	111

接收端收到每个码组后，先按式（6–2）～式（6–4）计算出S_1、S_2和S_3，再按表 6–2 判断错码情况。例如，若接收码组为 0000011，按式（6–2）～式（6–4）计算可得：S_1=0，S_2=1，S_3=1。由于$S_1S_2S_3$等于 011，故根据表 6–1 可知在a_3位有一错码。按上述方法构造的码称为汉明码。

现在再来讨论线性分组码的一般原理。上面已经提到，线性码是指信息位和监督位满足一组线性方程的码。式（6–5）就是这样一组线性方程的例子。现在将它改写成

$$\begin{cases} 1 \cdot a_6 + 1 \cdot a_6 + 1 \cdot a_6 + 0 \cdot a_6 + 1 \cdot a_6 + 0 \cdot a_6 + 0 \cdot a_0 = 0 \\ 1 \cdot a_6 + 1 \cdot a_6 + 0 \cdot a_6 + 1 \cdot a_6 + 0 \cdot a_6 + 1 \cdot a_6 + 0 \cdot a_0 = 0 \\ 1 \cdot a_6 + 0 \cdot a_6 + 1 \cdot a_6 + 1 \cdot a_6 + 0 \cdot a_6 + 0 \cdot a_6 + 1 \cdot a_0 = 0 \end{cases} \tag{6–7}$$

上式中已将“⊕”简写为“+”。在本章后面，除非另加说明，这类式中的“+”都指模 2 加。式（6–7）可以表示成如下矩阵形式：

$$\begin{bmatrix} 1110100 \\ 1101010 \\ 1011001 \end{bmatrix} \begin{bmatrix} a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \text{（模 2）} \tag{6–8}$$

上式还可以简记为

$$\boldsymbol{H}\cdot\boldsymbol{A}^{\mathrm{T}}=\boldsymbol{O}^{\mathrm{T}} \text{ 或 } \boldsymbol{A}\cdot\boldsymbol{H}^{\mathrm{T}}=O \tag{6-9}$$

其中

$$\boldsymbol{H}=\begin{bmatrix}1110100\\1101010\\1011001\end{bmatrix}，\quad \boldsymbol{A}=[a_6a_5a_4a_3a_2a_1a_0]，\quad \boldsymbol{O}=[000]。$$

右上标“T”表示将矩阵转置。

将$\boldsymbol{H}$称为监督矩阵。只要监督矩阵$\boldsymbol{H}$给定，编码时监督位和信息位的关系就完全确定了。由式（6−7）、式（6−8）都可看出，$\boldsymbol{H}$的行数就是监督关系式的数目，它等于监督位的数目r。$\boldsymbol{H}$的每行中“1”的位置表示相应码元之间存在的监督关系。例如$\boldsymbol{H}$的第一行 1110100 表示监督位a_2是由信息位$a_6a_5a_4$之和决定的。式（6−8）中的$\boldsymbol{H}$矩阵可以分成两部分，即

$$\boldsymbol{H}=\begin{bmatrix}1110\vdots100\\1101\vdots010\\1011\vdots001\end{bmatrix}=[\boldsymbol{PI}_r] \tag{6-10}$$

式中，$\boldsymbol{P}$为$r\times k$阶矩阵；$\boldsymbol{I}_r$为$r\times r$阶单位方阵，将具有$[\boldsymbol{PI}_r]$形式的$\boldsymbol{H}$矩阵称为典型监督矩阵。

由代数理论可知，$\boldsymbol{H}$矩阵的各行应该是线性无关的，否则将得不到r个线性无关的监督关系式，也就得不到r个独立的监督位。若一矩阵能写成典型监督矩阵形式［$\boldsymbol{PI}_r$］，则其各行一定是线性无关的。因为容易验证$[\boldsymbol{I}_r]$的各行是线性无关的，故［$\boldsymbol{PI}_r$］的各行也是线性无关的。

类似于将式（6−5）改写成式（6−8）中矩阵形式那样，式（6−14）也可以改写成矩阵形式

$$\begin{bmatrix}a_2\\a_1\\a_0\end{bmatrix}=\begin{bmatrix}1110\\1101\\1011\end{bmatrix}\begin{bmatrix}a_6\\a_5\\a_4\\a_3\end{bmatrix} \tag{6-11}$$

或者

$$\left[a_2a_1a_0\right]=\left[a_6a_5a_4a_3\right]\begin{bmatrix}111\\110\\101\\011\end{bmatrix}=\left[a_6a_5a_4a_3\right]\boldsymbol{Q} \tag{6-12}$$

式中，$\boldsymbol{Q}$ 为 $k\times r$ 阶矩阵，它为 $\boldsymbol{P}$ 的转置，即

$$\boldsymbol{Q}=\boldsymbol{P}^{\mathrm{T}} \tag{6-13}$$

式（6−12）表明，信息位给定后，用信息位的行矩阵乘以矩阵 $\boldsymbol{Q}$ 就产生出监督位。

将 $\boldsymbol{Q}$ 的左边加上一 $k\times k$ 阶单位方阵就构成一矩阵 $\boldsymbol{G}$

$$\boldsymbol{G}=\left[\boldsymbol{I}_k\boldsymbol{Q}\right]=\begin{bmatrix}1000111\\0100110\\0010101\\0001011\end{bmatrix} \tag{6-14}$$

$\boldsymbol{G}$ 称为生成矩阵，因为由它可以产生整个码组，即

$$[a_6 a_5 a_4 a_3 a_2 a_1 a_0] = [a_6 a_5 a_4 a_3] \cdot \boldsymbol{G} \tag{6–15}$$

或者

$$A = [a_6 a_5 a_4 a_3] \cdot \boldsymbol{G} \tag{6–16}$$

一般来说，式（6–16）中 $\boldsymbol{A}$ 为一 n 列的行矩阵。此矩阵的 n 个元素就是码组中的 n 个码元，所以发送的码组就是 $\boldsymbol{A}$。此码组在传输中可能由于干扰引入差错，故接收码组一般与 $\boldsymbol{A}$ 不一定相同。若设接收码组为一 n 列的行矩阵 $\boldsymbol{B}$，即

$$\boldsymbol{B} = \left[b_{n-1} b_{n-2} \cdots b_0\right] \tag{6–17}$$

则发送码组和接收码组之差为

$$\boldsymbol{B} - \boldsymbol{A} = \boldsymbol{E} \quad (\text{模 } 2) \tag{6–18}$$

它就是传输中产生的错码行矩阵

$$\boldsymbol{E} = \left[e_{n-1} e_{n-2} \cdots e_0\right] \tag{6–19}$$

其中

$$e_i = \begin{cases} 0, & b_i = a_i \\ 1, & b_i \neq a_i \end{cases}$$

因此，若e_i=0，则表示该位接收码元无错；若e_i=1，则表示该位接收码元有错。式（6–18）也可以改写成

$$\boldsymbol{B} = \boldsymbol{A} + \boldsymbol{E} \tag{6–20}$$

例如，若发送码组 $\boldsymbol{A}$=［1000111］，错码矩阵 $\boldsymbol{E}$=[0000100]，则接收码组 $\boldsymbol{B}$=[1000011]。错码矩阵有时也称为错误图样。

接收端译码时，可将接收码组 $\boldsymbol{B}$ 代入式（6–9）中计算。若接收码组中无错码，即 $\boldsymbol{E}$=0，则 $\boldsymbol{B}=\boldsymbol{A}+\boldsymbol{E}=\boldsymbol{A}$，把它代入式（6–9）后，该式仍成立，即

$$\boldsymbol{B} \cdot \boldsymbol{H}^{\mathrm{T}} = \boldsymbol{O} \tag{6–21}$$

当接收码组有错时，$\boldsymbol{E} \neq 0$，将 $\boldsymbol{B}$ 代入式（6–9）后，该式不一定成立。在错码较多，已超过这种编码的检错能力时，$\boldsymbol{B}$ 变为另一许用码组，则式（6–21）仍能成立。这样的错码是不可检测的。在未超过检错能力时，上式不成立，即其右端不等于零。假设这时式（6–21）的右端为 S，即

$$\boldsymbol{B} \cdot \boldsymbol{H}^{\mathrm{T}} = \boldsymbol{S} \tag{6–22}$$

将 $\boldsymbol{B}=\boldsymbol{A}+\boldsymbol{E}$ 代入式（6–22）中，可得

$$\boldsymbol{S} = (\boldsymbol{A} + \boldsymbol{E}) \cdot \boldsymbol{H}^{\mathrm{T}} = \boldsymbol{A} \cdot \boldsymbol{H}^{\mathrm{T}} + \boldsymbol{E} \cdot \boldsymbol{H}^{\mathrm{T}}$$

由式（6–9）可知

$$\boldsymbol{A} \cdot \boldsymbol{H}^{\mathrm{T}} = 0$$

所以

$$\boldsymbol{S} = \boldsymbol{E} \cdot \boldsymbol{H}^{\mathrm{T}} \tag{6–23}$$

式中，$\boldsymbol{S}$ 称为校正子，有可能利用它来指示错码位置。这一点可以直接从式（6–23）中看出，$\boldsymbol{S}$ 只与 $\boldsymbol{E}$ 有关，而与 $\boldsymbol{A}$ 无关，这就意味着 $\boldsymbol{S}$ 与错码 $\boldsymbol{E}$ 之间有确定的线性变换关系。若 $\boldsymbol{S}$

和 ***E*** 之间一一对应，则 ***S*** 将能代表错码的位置。

6.2.2 （7，4）汉明码的 MATLAB 实现

【例 6-1】 找出（7，4）汉明码的所有码字，并证明其最小码距为 3。

解：构造（15，11）汉明码的校验矩阵为

$$\boldsymbol{H}=\begin{bmatrix}1110100\\1101010\\1011001\end{bmatrix}$$

其相应的生成矩阵为

$$\boldsymbol{G}=\begin{bmatrix}1000111\\0100110\\0010101\\0001011\end{bmatrix}$$

可以知道汉明码有 2^4=16 个码字，码长为 7，编码速率为 4/7=0.57。线性分组码的最小码距为所有码字中非零码字的码重，通过这个特性，可以利用MATLAB编程来求得其最小码距。

程序如下：

```
echo on
k=4;
for i=1:2^k
   for j=k:-1:1
        if rem(i-1,2^(-j+k+1)) >= 2^(-j+k)
             u(i,j) = 1;
        else
             u(i,j) = 0;
        end
   end
end

g = [1 0 0 0 1 1 1
     0 1 0 0 1 1 0
     0 0 1 0 1 0 1
     0 0 0 1 0 1 1];
c = rem(u*g,2);
w_min=min(sum((c(2:2^k,:))’))
```

【例 6-2】 对于上述的（7，4）汉明码，当接收码元为 *b*=[1 0 1 0 1 1 1]时，请用 MATLAB 编程进行检错和纠错。

解：根据汉明码的原理，接收码组和校验矩阵的乘积应该为一个 O 矩阵（即元素全为 0 的矩阵，否则就有错误。根据汉明码最小码距为 3 的特性，汉明码可以纠正一个错误，如超过这个限度则不能纠正。在编程中首先根据生成矩阵得到监督矩阵，然后用监督矩阵和接收

码组相乘，根据结果就可以进行纠错了。

```
b=[1 0 1 0 1 1 1]
g = [1 0 0 0 1 1 1
     0 1 0 0 1 1 0
     0 0 1 0 1 0 1
     0 0 0 1 0 1 1];

h=[g(:,5:7)' eye(3,3)]
mod(b*h',2)
```

程序运行的结果为：1 0 1，则根据表 6−2 可以得到第三个码元错误，所以原来的码组应该为 1 0 0 0 1 1 1。

6.3 卷积码原理及其实现

6.3.1 卷积码的基本原理

卷积码又称连环码，是 1955 年提出来的一种纠错码，它和分组码有明显的区别。（n,k）线性分组码中，本组 $r = n-k$ 个监督元仅与本组 k 个信息元有关，与其他各组无关，也就是说分组码编码器本身并无记忆性。卷积码则不同，每个（n,k）码段（也称子码），通常较短的 n 个码元不仅与该码段内的信息元有关，而且与前面 m 段的信息元有关。通常称 m 为编码存储。卷积码常用符号（n，k，m）表示。图 6−2 是卷积码（2，1，2）的编码器。它由移位寄存器、模二加法器及开关电路组成。

起始状态，各级移位寄存器清零，即$S_1S_2S_3$为 000。S_1等于当前输入数据，而移位寄存器状态S_2S_3存储以前的数据，输出码字C由下式确定：

$$\begin{cases} C_1 = S_1 \oplus S_2 \oplus S_3 \\ C_2 = S_1 \oplus S_3 \end{cases} \tag{6−24}$$

当输入数据 D=[11010]时，输出码字可以计算出来，具体计算过程如表 6−3 所示。另外，为了保证全部数据通过寄存器，还必须在数据位后加 3 个 0。

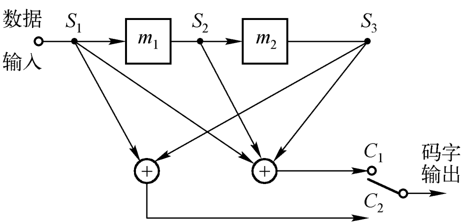


图 6−2 卷积码（2，1，2）编码器

从上述的计算可知，每 1 位数据，影响（m+1）个输出子码，称（m+1）为编码约束度。每个子码有 n 个码元，在卷积码中有约束关系的最大码元长度则为（m+1）n，称为编码约束长度。（2，1，2）卷积码的编码约束度为 3，约束长度为 6。

表 6-3 （2，1，2）编码器的工作过程

S_1	1	1	0	1	0	0	0	0
S_3S_2	00	01	11	10	01	10	00	00
C_1C_2	11	01	01	00	10	11	00	00
状态	*a*	*b*	*d*	*c*	*b*	*c*	*a*	*a*

卷积码同样也可以用矩阵的方法描述，但较抽象。因此，常采用图解的方法直观描述其编码过程。常用的图解法有 3 种：状态图、树图和格图。

1．状态图

图 6-3 是（2,1,2）卷积编码器的状态图。在图中有 4 个节点*a*、*b*、*c*、*d*，同样分别表示S_3S_2的 4 种可能的状态：00、01、10 和 11。每个节点有两条线离开该节点，实线表示输入数据为 0，虚线表示输入数据为 1，线旁的数字即为输出码字。

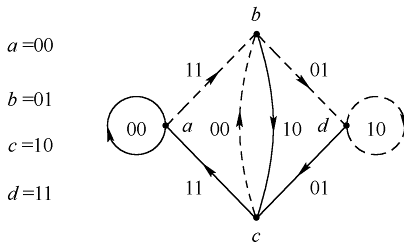


图 6-3 （2,1,2）码的状态图

2．树图

树图描述的是在任何数据序列输入时，码字所有可能的输出。对应于图 6-4 所示的（2,1,2）卷积码的编码电路，可以画出其树图如图 6-4 所示。

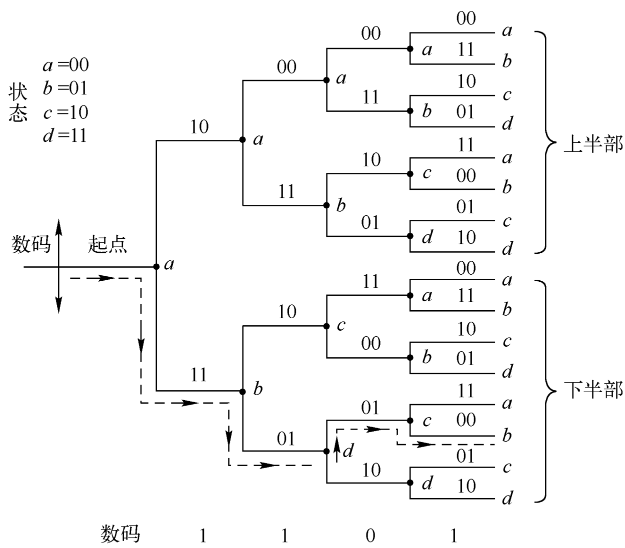


图 6-4 （2，1，2）码的树图

以$S_1S_2S_3$=000 作为起点。若第一位数据S_1=0，输出C_1C_2=00，从起点通过上支路到达状态a，即S_1S_2=00；若S_1=1，输出C_1C_2=11，从起点通过下支路到达状态b，即S_3S_2=01；依此类推，可得整个树图。输入不同的信息序列，编码器就走不同的路径，输出不同的码序列。例如当输入数据为[11010]时，其路径如图中虚线所示，并得到输出码序列为[11010100…]，与表 6–3 的结果一致。

3．格状图

格状图也称网络图或篱笆图，它由状态图在时间上展开而得到，如图 6–5 所示。图中画出了所有可能数据输入时，状态转移的全部可能轨迹，实线表示数据为 0，虚线表示数据为 1，线旁数字为输出码字，节点表示状态。

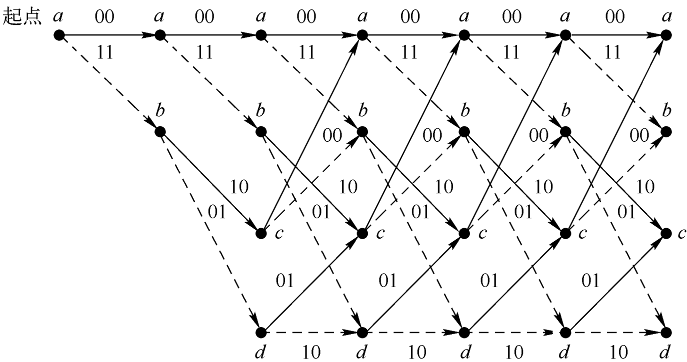


图 6–5 （2，1，2）码的格状图

以上 3 种卷积码的描述方法，不但有助于求解输出码字，了解编码工作过程，而且对研究解码方法也很有用。

卷积码的译码可分为代数译码和概率译码两大类。代数译码是利用生成矩阵和监督矩阵来译码，最主要的方法是代数逻辑译码。概率译码比较实用的有两种：维特比译码和序列译码。目前，概率译码已成为卷积码最主要的译码方法。本节将简要讨论维特比译码。

维特比译码是一种最大似然译码算法。最大似然译码算法的基本思路是，把接收码字与所有可能的码字比较，选择一种码距最小的码字作为解码输出。由于接收序列通常很长，所以维特比译码时做了简化，即它把接收码字分段处理。每接收一段码字，计算、比较一次，保留码距最小的路径，直至译完整个序列。

现以上述（2，1，2）码为例说明维特比译码过程。设发送端的信息数据 D=[11010000]，由编码器输出的码字 C=[1101010010110000]，接收端接收的码序列 B=[0101011010010010]，有 4 位码元差错。下面参照图 6–5 的格状图说明译码过程。

如图 6–6 所示，先选前 3 个码作为标准，对到达第 3 级的 4 个节点的 8 条路径进行比较，逐步算出每条路径与接收码字之间的累计码距。累计码距分别用括号内的数字标出，对照后保留一条到达该节点的码距较小的路径作为幸存路径。再将当前节点移到第 4 级，计算、比较、保留幸存路径，直至最后得到到达终点的一条幸存路径，即为解码路径，如图 6–6 中实线所示。根据该路径，得到解码结果。

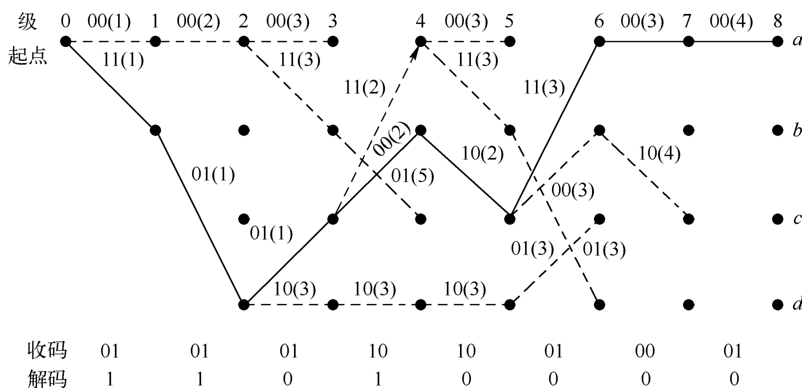


图 6-6　维特比译码格图

6.3.2　卷积码编译码的 MATLAB 实现

MATLAB 提供了卷积码编译码的函数 convenc()，可以快速地得到编译码的结果，其中的详细说明可以参考 MATLAB 的帮助文件。

涉及卷积码编码的函数主要有以下三个函数：

（1）code = convenc(msg.trellis)

（2）code = convenc(msg.trellis,puncpat)

其中 puncpat 定义了打孔矩阵的格式。

（3）code = convenc(msg,trellis,…,init_state)

其中 init_state 为编码寄存器的初始状态

涉及卷积码译码的函数为

decoded = vitedec(code,trellis,tblen,opmode,dectype)

其中，code 为输入的待译码信息，trellis 为卷积码编码器结构，tblen 表示译码深度，opmode 表示译码器的操作模式，dectype 给出译码器判决的类型（软判决或硬判决）。

【例 6−3】 使用 MATLAB 实现卷积码的编译码。信源为随机产生数，并进行软判决译码。

```
msg = randint(4000,1,2,139);                                  %产生随机数
t = poly2trellis(7,[171,133]);                                %定义编码器结构
code = convenc(msg,t);                                        %进行卷积码编码
ncode = awgn(code,6,'measured',244);                          %加入噪声
qcode = quantize(ncode,[0.001,.1,.3,.5,.7,.9,0.999]);         %进行量化，以便进行软判决
tblen = 48; delay = tblen;                                    %译码深度
decoded = vitdec(qcode,t,tblen,'cont','soft',3);              %维特比译码
[number,ratio] = biterr(decoded(delay+1:end),msg(1:den-delay)) %计算误码率
```

仿真得到的误码率为 1.3×10^{-4}。

6.3.3　卷积码的 dsPIC 实现

1．卷积编码

在文件 PSKTables.h 中，定义了一个卷积码生成，在表格中，把 5 个比特（输入 1 个比

特，存储 4 个比特）涉及的所有情况罗列出来，按照每个输入对应一个输出的形式写成表格。在使用的时候直接用 1 个输入比特和 4 个存储比特来查表，得到输出的两个比特。

```
/* For the QPSK modulator/demodulator, rate 1/2 constraint length 5          */
/*      convolutional FEC coding is used.                                     */
/* The generator polynomials used are:                                        */
/*    g1(x) = x^4 + x^3             + 1        = 0x19                         */
/*    g0(x) = x^4   +        x^2 + x + 1       = 0x17                         */
/*                                                                            */
/*                                g1(x)                                       */
/*              /----+--------+-------------------------+                     */
/*             /     |        |                         |                     */
/* symbol msb      ---      ---      ---      ---      ---                    */
/*                | b4|<---| b3|<---| b2|<---| b1|<---| b0| <-invrtd data in  */
/* symbol lsb     ---      ---      ---      ---      ---                     */
/*             \     |                 |        |        |                    */
/*              \----+-----------------+--------+--------+                    */
/*                                g0(x)                                       */
/*                                                                            */
/* Lookup table to get symbol from non-inverted data stream                   */

const char ConvolutionCodeTable[32] =
{
        2, 1, 3, 0, 3, 0, 2, 1,
        0, 3, 1, 2, 1, 2, 0, 3,
        1, 2, 0, 3, 0, 3, 1, 2,
        3, 0, 2, 1, 2, 1, 3, 0
};
```

在 PSKMod.h 文件中，进行卷积码编码的代码是：

```
symb = ConvolutionCodeTable[m_TxShiftReg&0x1F];     /*get next convolution code */
```

其中，m_TxShiftReg 用来存储 1 个输入比特和 4 个存储比特，然后根据这 5 个比特查 ConvolutionCodeTable 表得到两个输出比特。这里的两个输出比特是放在一起存储的。当查表结果为 3 时，输出为 11；查表结果为 2 时，输出为 10，查表结果为 1 时，输出为 01；查表结果为 0 时，输出为 00。

2．卷积译码

当采用 QPSK 通信方式时，发送信号采用卷积编码，接收程序利用维特比译码方法对接收到的信息进行卷积码译码。

在 PskDet.c 文件中，进行卷积码译码的程序如下：

```
/*////////////////////////////////////////////////////////////////////    */
/* Soft-decision Viterbi decoder function.                    */
/* ////////////////////////////////////////////////////////////////////    /
int ViterbiDecode( double newangle)
```

```
{
double pathdist[32];
double min;
int bitestimates[32];
long ones;
int i;
const double* pAngleTbl;
        min = 1.0e37;                       /*确保可以找到最小值 */
        if( newangle >= PI2/2 )             /*处理 +/- 2PI 相位模糊*/
                pAngleTbl = ANGLE_TBL2 ;    /*采用两个不同的表 */
        else
                pAngleTbl = ANGLE_TBL1;
        for(i = 0; i < 32; i++)             /*计算所有可能的距离 */
        {                                   /* 'i'的 lsb 是最后估计出的比特 */
                pathdist[i] = m_SurvivorStates[i / 2].Pathdistance +
                                fabs(newangle - pAngleTbl[ ConvolutionCodeTable[i] ]);
                if(pathdist[i] < min)       /* 跟踪最小距离 */
                        min = pathdist[i];
                /* 移入最新估计的比特 */
                bitestimates[i] = ((m_SurvivorStates[i / 2].BitEstimates) << 1) + (i & 1);
        }
        for(i = 0; i < 16; i++)             /*比较进入同一状态的路径长度，只保留最小的路径
到 m_SurvivorStates[]*/

        {
                if(pathdist[i] < pathdist[16 + i])
                {
                        m_SurvivorStates[i].Pathdistance = pathdist[i] - min;
                        m_SurvivorStates[i].BitEstimates = bitestimates[i];
                }
                else
                {
                        m_SurvivorStates[i].Pathdistance = pathdist[16 + i] - min;
                        m_SurvivorStates[i].BitEstimates = bitestimates[16 + i];
                }
        }
        ones = 0;
        for(i = 0; i < 16; i++)             /* 判断在比特 20 的位置是否有超过一个的 0 */
                ones += (m_SurvivorStates[i].BitEstimates&(1L << 20));
        if( ones == (8L << 20 ) )
                return ( rand() & 0x1000 );     /*只能猜了 */
        else
                return(ones > (8L << 20) );     /*这是可以输出最可能的比特 */
}
```

第 7 章　基于 dsPIC 无线通信设备 NUE–PSK31 型数字调制解调器实例剖析

7.1　PSK31 型数字调制解调器简介

dsPIC33F 系列数字信号控制器在无线通信中有着广泛的应用。这里以一个业余无线电中使用的 NUE-PSK3.1 型数字调制解调器为例，介绍用 dsPIC33F 实现无线数字通信的相关功能的方法。

NUE-PSK3.1 型数字调制解调器是一款低速率、低功耗的短波无线数字调制解调器，它采用 BPSK 和 QPSK 调制方式，在 2kHz 的音频带宽内以 31.25bit/s 的速率传输字符信息。NUE-PSK3.1 型数字调制解调器有一个简单的人机接口，字符输入是一个标准的 PS/2 接口，可以连接标准的 PS/2 接口键盘；输出显示采用液晶显示器（LCD），显示区可分为上、下两个部分，上部分以图形的方式显示接收信号的频谱幅度，下部分显示收发的字符信息。业余无线电爱好者将该设备的音频输入输出接口与低功率 SSB（单边带）电台的音频接口相连，可以不用计算机，直接通过键盘和 LCD 与远在异地的同伴进行信息交流，NUE-PSK3.1 型数字调制解调器的实物如图 7−1 所示。

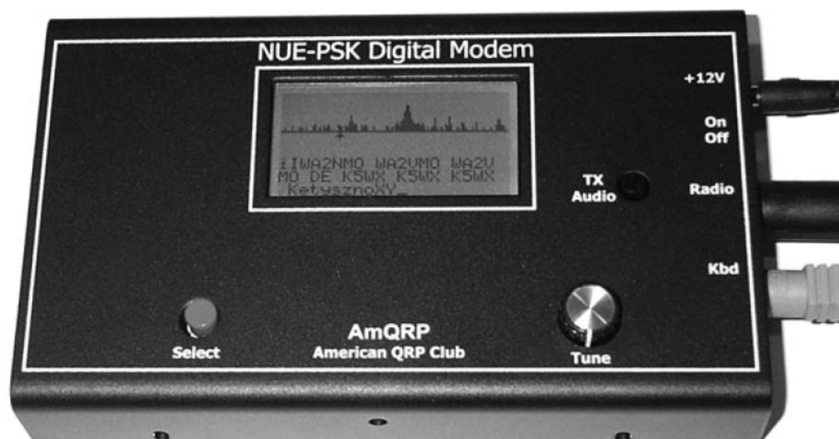


图 7−1　NUE-PSK3.1 型数字调制解调器

从功能上讲，NUE-PSK3.1 型数字调制解调器有以下几个特点：

- 可独立工作的、半双工的数字调制解调器。
- 小巧的手持式设备，无需计算机的介入。

- 与 SSB 电台之间通过音频 I/O 接口相连。
- 设备自带频谱显示和字符显示 LCD。
- 调制方式为 BPSK 和 QPSK。
- 通过 LCD 的菜单进行参数配置。
- 具有 PS/2 接口，可外接标准键盘。
- 可采用内部电池供电。
- 开放源代码，源代码采用 C 语言编写。

从硬件组成上看，NUE-PSK3.1 的核心器件是一片dsPIC33FJ128MC706 数字信号控制器，系统时钟采用外接 10MHz晶振产生主振荡器时钟源。外部扩展了人机交互接口，包括PS/2 键盘接口、LCD接口；外部存储器接口通过I^2C接口连接一片EEPROM芯片；模拟信号电路通过一个高速同步串行接口（SPI）连接一片 12bit D/A转换芯片用于模拟信号输出，通过另外一个SPI连接一片可编程增益放大器。

NUE-PSK3.1 型调制解调器的硬件原理图如图 7−2 所示。

NUE-PSK3.1 的软件主要是基于 dsPIC33 数字信号控制器程序实现的，包含如下几个模块：

- 键盘输入模块：接收来自键盘输入的字符。
- LCD 显示模块：显示收发字符和接收信号的频谱幅度。
- 调制模块：发送信号调制。
- 解调模块：接收信号解调。
- 参数配置模块等。

NUE-PSK3.1 调制解调器采用外接 12V 供电，内部的电源转换电路将输入的 12V 电源转换成系统所需要的 5V 和 3.3V 电源。

NUE-PSK3.1 型调制解调器主要器件功能如表 7−1 所示。

表 7−1　NUE-PSK3.1 型数字调制解调器主要器件功能

器 件 编 号	器 件 型 号	功 能 描 述
U1	dsPIC33FJ128MC706	数字信号控制器
U2、U3	TXB0108	8 位电平转换器
U4	24AA256	EEPROM 存储器
U5	MC68HC908QY4	8 位微处理器，用于扫描键盘输入
U6	MCP4922	8 位 D/A 转换器
U7	MCP6S21	可编程模拟增益放大器
U8	MCP601	运算放大器
U9	PT78ST105H	5V 电源稳压器
U10	LP2950	3.3V 电源生成器
LCD	CFAG12864	128×64 像素的 LCD
ENC-1	P10860	旋转编码器

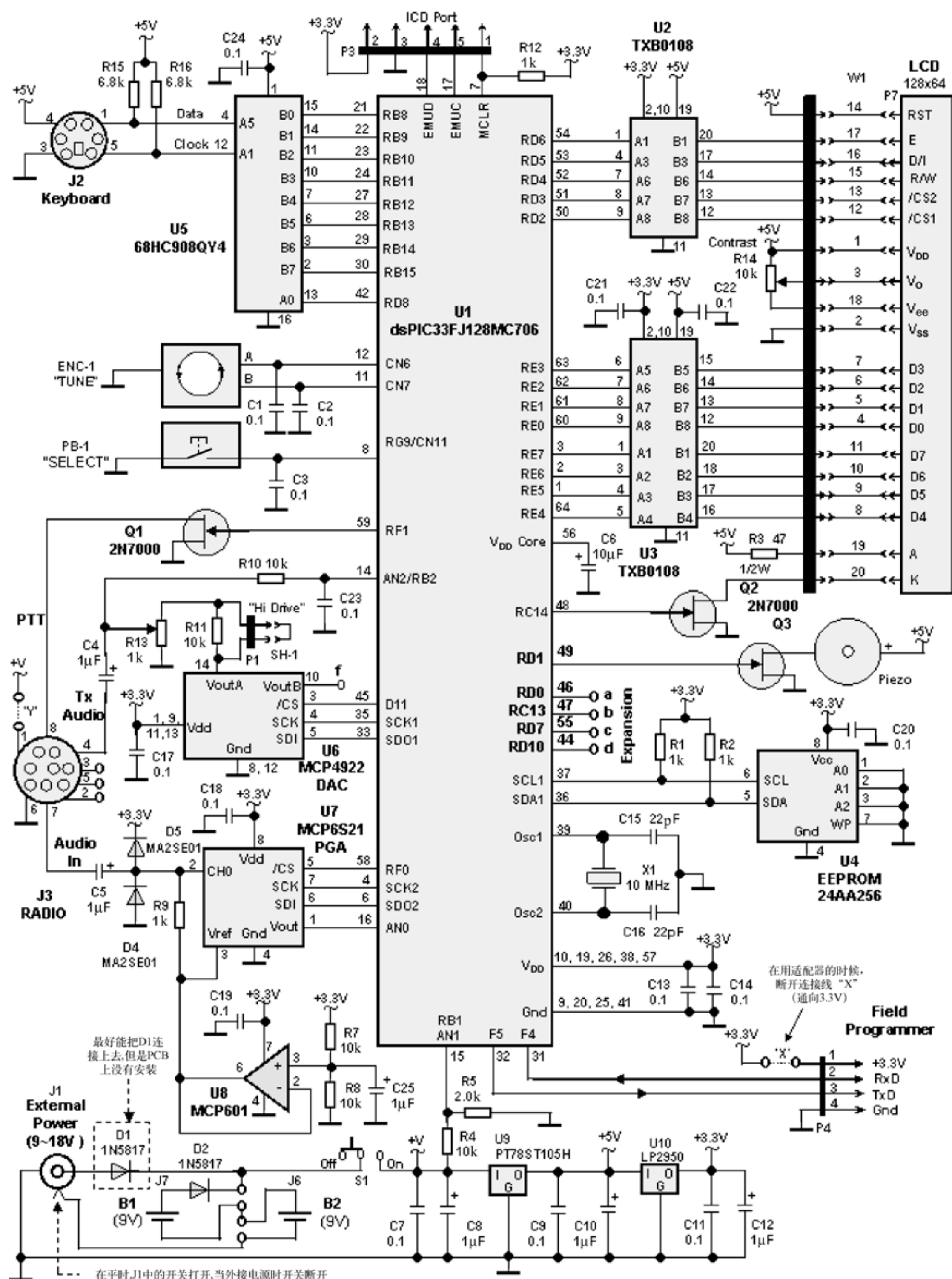


图 7-2　NUE-PSK3.1 型数字调制解调器的硬件原理图

NUE-PSK3.1 型调制解调器工作时与外设的连接示意图如图 7−2 所示。



图 7−3　NUE-PSK3.1 型数字调制解调器外部连接示意图

dsPIC33FJ128MC706 数字信号控制器内部集成了 128KB 的闪存，其引脚数量为 64，并且该控制器是电动机控制系列处理器，内部资源十分丰富。

NUE-PSK3.1 型调制解调器工作时需要连接 12V 电源、PS/2 接口键盘和具有音频输入接口的低功率 SSB（单边带）电台。

NUE-PSK 开机后 LCD 上半部会实时显示图形方式的带宽内信号的频谱幅度，信号显示部分有一个游标，用户可以通过 NUE-PSK 上的调谐旋钮或键盘上的上、下箭头按键移动这个游标的位置，移动游标可以移动并锁定 PSK 信号，同时进行解码，将接收的 ASCII 码显示在 LCD 的下半部分。当用户需要发送信息时，按下〈F12〉键进入发送模式，然后通过键盘敲入字符信息，输入的字符会显示在 LCD 上并发送出去，发送完毕再按下〈F12〉键进入接收模式等待对方的应答。

7.2　PSK31 型数字调制解调器人机交互接口

人机交互接口是用户与 NUE-PSK3.1 之间建立联系、交流信息的接口。NUE-PSK3.1 调制解调器的人机交互接口主要由 PS/2 键盘输入接口、旋转编码器输入接口和 LCD 液晶显示接口组成。PS/2 键盘输入接口将用户敲击键盘的字符信息传递给 CPU，选择编码器采集用户选择频率的信息，LCD 显示接收信号的频率、幅度以及用户输入与输出的信息等。

7.2.1　PS/2 键盘输入接口

NUE-PSK3.1 型调制解调器的键盘输入接口采用一片 MC68HC908 单片机实现。MC68HC908 是 Motorola 公司推出的以 Flash 作为存储器的 8 位单片机，该处理器的最大特点是提供了多种外围接口模块，其中之一就是键盘中断模块（KBI）。MC68HC908 提供的键盘接口采用标准的 PS/2 接口，该接口多用于键盘和鼠标等外设的连接。PS/2 接口的电气图形符号如图 7−4 所示。

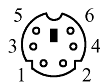


图 7−4　PS/2 标准接口的电气图形符号

PS/2 标准接口的引脚定义见表 7−2。

表 7−2　PS/2 标准接口的引脚定义

引　脚　号	描　　述
1	数据
2	未定义
3	地
4	电源+5V
5	时钟
6	未定义

PS/2 协议是一种双向同步串行通信协议。通信双方通过时钟引脚同步，并通过数据引脚交换数据。任何一方如果想终止另外一方通信时，只需要把时钟引脚拉到低电平即可。通常主机可以终止通信而 PS/2 设备不会终止主机的数据发送。PS/2 接口的设备间传输数据的最大时钟频率是 33kHz，推荐值在 15kHz 左右。传输数据格式遵循 11 位串行协议，每个比特占用一个时钟周期，其波形如图 7−5 所示。

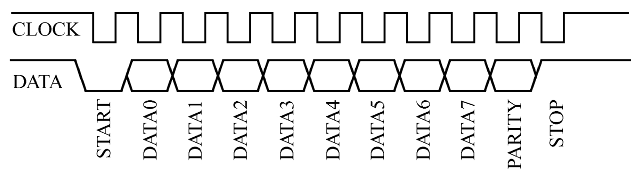


图 7−5　PS/2 标准接口数据传输波形

其中第一个比特为起始位，总为低电平；后面跟着 8 位数据，低位在前，高位在后；之后是一个比特的校验位，这里采用奇校验；最后一个比特是停止位，总为高电平。当用户按下或者释放键盘按键时，键盘都会通过 PS/2 标准接口发送数据，这个数据通过“扫描码”的格式发送到主机中。扫描码有两种类型：“通码”和“断码”，当一个键被按下时发送“通码”；当一个键被释放时发送“断码”。每个按键都被分配了唯一的“通码”和“断码”，这样主机通过查找唯一的扫描码就可以判断是哪个按键。每个键一整套的“通断码”组成了扫描码集，目前有三套扫描码集，常用键盘采用第二套扫描码集。

MC68HC908 单片机将 PS/2 键盘上的串行扫描码转换成并行数据，通过中断方式通知 dsPIC33FJ128MC706。dsPIC33FJ128MC706 使用了 PORT B 的高 8 位（RB8～RB15）作为键盘扫描码的输入端，使用 PORT D 的第 8 位（RD8）作为中断引脚，当 MC68HC908 接收到键盘数据后，通过 RD8 向 dsPIC33FJ128MC706 发送中断请求，dsPIC33FJ128MC706 响应此中断，在中断处理程序中读取 RB8～RB15 上的数据，获得键盘输入的扫描码。

在源文件 NUE_PSK_main_1.c 中，程序调用 init_Key_isr ()函数进行键盘输入中断的初始化，该函数在 NUE_PSK_Keyboard.c 文件中定义为

```
void init_Key_isr ( void )     {

     Key_Strobe_I =  1;     /*    键盘选通    */
     Key_EP       =  0;     /*    选通上升沿中断    */
```

```
Key_IP      = 4;   /*  设置中断优先级  */
Key_IF      = 0;   /*  在使能前清除中断标志  */
Key_IE      = 1;   /*  使能键盘中断  */
Key_0_tris  = 1;   /*  键盘数据 0 输入  */
Key_1_tris  = 1;   /*  键盘数据 1 输入  */
Key_2_tris  = 1;   /*  键盘数据 2 输入  */
Key_3_tris  = 1;   /*  键盘数据 3 输入  */
Key_4_tris  = 1;   /*  键盘数据 4 输入  */
Key_5_tris  = 1;   /*  键盘数据 5 输入  */
Key_6_tris  = 1;   /*  键盘数据 6 输入  */
Key_7_tris  = 1;   /*  键盘数据 7 输入  */

}
```

其中，Key_Strobe_I 为中断输入信号，程序中将其定义为

```
#define    Key_Strobe_I    TRISDbits.TRISD8
```

键盘输入的中断服务例程为函数 key_isr（ void ），该函数将输入的扫描码转换成其他程序使用的 ASCII 码。

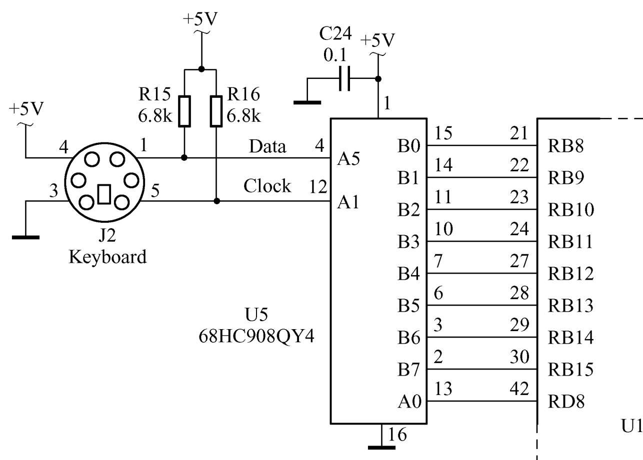


图 7-6　NUE-PSK3.1 的键盘接口设计

7.2.2　旋转编码器输入接口

旋转编码器是用来测量转速或旋转圈数的装置，常见的旋转编码器有磁旋转编码器和光电式旋转编码器两种。旋转编码器通过光电或磁电转换，可将输出轴的角位移、角速度等机械量转换成相应的电脉冲以数字量输出（REP）。它分为单路输出和双路输出两种。技术参数主要有每转脉冲数（几十个到几千个都有）和供电电压等。单路输出是指旋转编码器的输出是一组脉冲，而双路输出的旋转编码器输出两组 A/B 相位差 90°的脉冲，通过这两组脉冲不仅可以测量转速，还可以判断旋转的方向。NUE-PSK3.1 调制解调器使用旋转编码器为双路输出，它接收用户调整游标的位置信息，用户通过旋动旋转编码器旋钮选择 PSK 信号

的频点，也可以在配置界面中选择不同的功能选项。旋转编码器实物如图 7-7 所示。

dsPIC33FJ128MC706 通过内部集成的正交编码接口模块（QEI）接收旋转编码器的信号。其接口电路如图 7-8 所示。



图 7-7　旋转编码器实物图

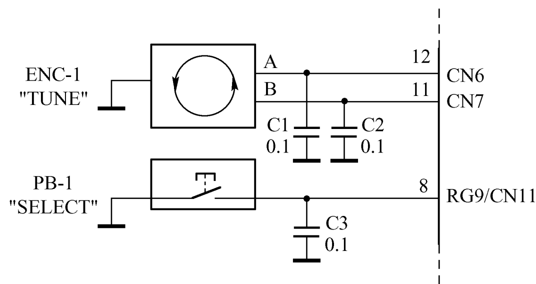


图 7-8　旋转编码器接口电路

正交编码接口模块（QEI）内部有一个 16 位基准时钟，可以接收从旋转编码器的 A、B 端子发送的脉冲信号，其中 A 连接到 QEA 引脚（与 CN6 复用）、B 连接到 QEB 引脚（与 CN7 复用）。如果 A 相（QEA）超前 B 相（QEB）那么旋转方向被认为是正向，反之为反向。QEA 和 QEB 引脚输入的每个脉冲作为增量脉冲计数器信号，每个 QEA 和 QEB 信号沿到来时，POSCNT 寄存器都会增量或者减量计数。

在源文件 NUE_PSK_main_1.c 中，程序调用 Init_QED ()函数进行正交编码接口模块的初始化，该函数在 NUE_PSK_QED.c 文件中定义为

```
void  Init_QED ( void ) {

    QEICONbits.UPDN_SRC     =    0;
    Nop()
    QEICONbits.TQCS         =    0;
    Nop()
    QEICONbits.POSRES       =    0;
    Nop()
    QEICONbits.TQCKPS       =    0;
    Nop()
    QEICONbits.TQGATE       =    0;
    Nop()
    QEICONbits.PCDOUT       =    0;
    Nop()
    QEICONbits.SWPAB        =    0;
    Nop()
    QEICONbits.QEIM         =    7;
    Nop()
    QEICONbits.QEISIDL      =    0;
    Nop()
    QEICONbits.CNTERR       =    0;
```

```
DFLTCONbits.QECK        =   7;
Nop()
DFLTCONbits.QEOUT       =   1;
Nop()
DFLTCONbits.CEID        =   1;
Nop()
DFLTCONbits.IMV         =   0;

POSCNT                  =   100;

MAXCNT                  =   0xFFFF;

Button1_tris            =   1;   /* as input */

CNPU1bits.CN11PUE       =   1; /* Button1 */
Nop()
CNPU1bits.CN6PUE        =   1; /* QED_1A */
Nop()
CNPU1bits.CN7PUE        =   1; /* QED_1B */

IFS3bits.QEIIF          =   0;
/*   don't enable QE interrupts */
IEC3bits.QEIIE          =   0;    /* set to 1 to enable QE interrupts */

}
```

其中，QEICONbits.QEIM=7 表示将 QEI 设置为 X4 模式，并且与 MAXCNT 寄存器相匹配时将位置计数器复位，这里 MAXCNT 寄存器被设置为 0xFFFF。

读取 QEI 计数器数值的代码在 Proc_QEI()函数中，该函数大约每 50ms 被执行一次。Proc_QEI()函数在文件 NUE_PSK_Subroutines_2C.c 中定义。

7.2.3 LCD 显示接口

NUE-PSK3.1 的显示器选用了一款具有图形显示功能的液晶显示屏。该显示屏可显示 128 行×64 列像素的图形或字符，显示信息为单色无灰度。12864LCD 的实物如图 7−9 所示。



图 7−9　12864LCD 的实物图

12864LCD 采用并行总线接口，引脚定义见表 7−3。

表 7−3　12864LCD 接口定义

引　脚　号	描　　述
1	电源+5V（V_{DD}）
2	地
3	对比度调整电压输入
4～11	数据线
12	片选 1（/CS1）
13	片选 2（/CS2）
14	复位
15	读/写控制
16	数据/指令控制
17	使能信号
18	负电压输出
19	LED 背光灯正极
20	LED 背光灯负极

12864LCD 的引脚时序如图 7−10 所示。对 12864LCD 的控制分为指令和数据两部分，通过 D/I 引脚控制数据总线上的信息含义，当该引脚为高时数据总线上为数据信息，当该引脚为低时数据总线上为指令信息。指令信息包括 12864LCD 的开启与关闭、设置行或列的地址、设置起始位置以及读取 12864LCD 状态信息等。

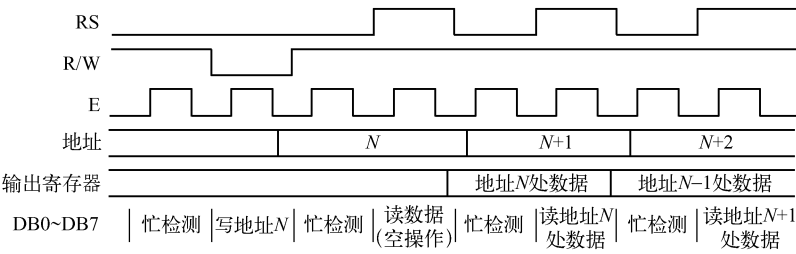


图 7−10　12864LCD 的引脚时序图

dsPIC33FJ128MC706 采用 PORT E（RE7～RE0）的 8 位端口作为与 12864LCD 相连的数据总线接口，采用 PORT D（RD6～RD2）作为控制引脚。dsPIC33FJ128MC706 与 12864LCD 之间用 TXB0108 进行电平转换。TXB0108 是 8 位双向电平转换器，其端口 A 适用的电平为 1.2～3.6V，端口 B 适用的电平为 1.65～5.5V。dsPIC33FJ128MC706 还使用了 RC14 引脚作为 12864LCD 开启背光灯的控制端。12864LCD 接口电路如图 7−11 所示。

对 12864LCD 显示屏的初始化是通过源文件 NUE_PSK_main_1.c 中的 Init_LCD ()函数完成的，该函数在 NUE_PSK_LCD_single_1.c 文件中定义，具体代码如下：

```
void Init_LCD ( void )  {

        LCD_D0      =      0;
```

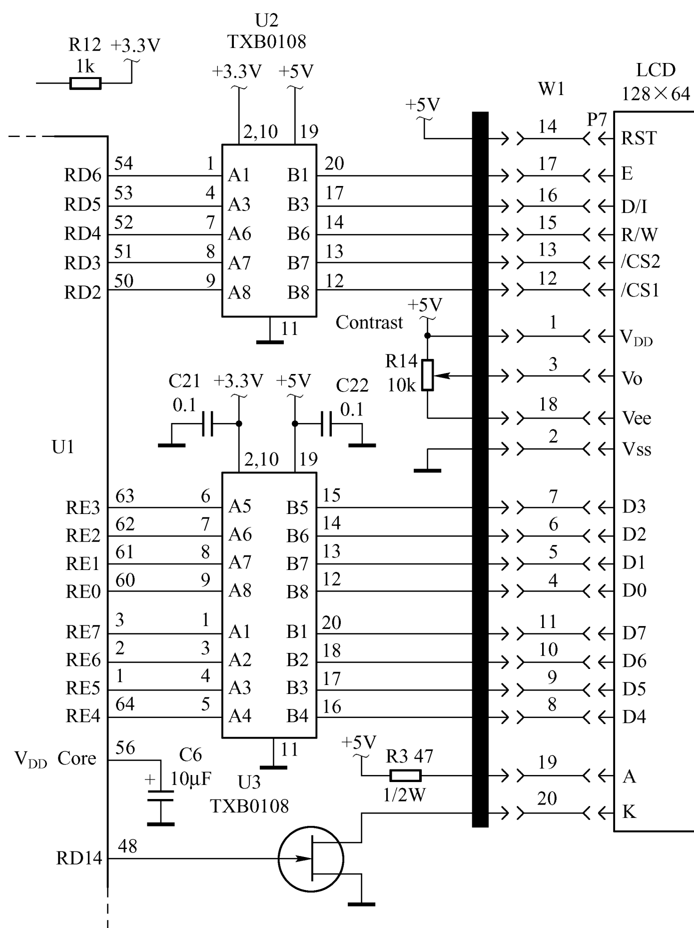


图 7-11　12864LCD 接口电路图

```
LCD_D1      =    0;
LCD_D2      =    0;
LCD_D3      =    0;
LCD_D4      =    0;
LCD_D5      =    0;
LCD_D6      =    0;
LCD_D7      =    0;
LCD_CS1     =    1;
LCD_CS2     =    1;
LCD_RW      =    1;
LCD_DI      =    1;
LCD_E =     0;

BackLight_tris = 0;

TRISEbits.TRISE0 =    0;   /*   LCD Data 0     */
TRISEbits.TRISE1 =    0;   /*   LCD Data 1     */
```

```
TRISEbits.TRISE2 =      0;     /*     LCD Data 2        */
TRISEbits.TRISE3 =      0;     /*     LCD Data 3        */
TRISEbits.TRISE4 =      0;     /*     LCD Data 4        */
TRISEbits.TRISE5 =      0;     /*     LCD Data 5        */
TRISEbits.TRISE6 =      0;     /*     LCD Data 6        */
TRISEbits.TRISE7 =      0;     /*     LCD Data 7        */

                                      /*   vers 9    -> vers 11    */
TRISDbits.TRISD0=       0;     /*     LCD CS1 -> Spare        */
TRISDbits.TRISD1=       0;     /*     LCD CS2 -> Piezo        */
TRISDbits.TRISD2=       0;     /*     LCD RW -> LCD CS1  */
TRISDbits.TRISD3=       0;     /*     LCD DI -> LCD CS2   */
TRISDbits.TRISD4=       0;     /*     LCD E -> LCD RW      */
TRISDbits.TRISD5=       0;     /*     unused -> LCD DI        */
TRISDbits.TRISD6=       0;     /*     unused -> LCD E          */


Write_lcd_cntrl_regs (BOTH,0x3F);           /* Turn display on */

Clear_Display ( 0 );                  /* Clear the single graphic display */
BackLight = 1;               /* Turn on the backlight */
}
```

其中，Write_lcd_cntrl_regs (BOTH,0x3F)函数是开启 LCD 的显示功能；Clear_Display (0)是将屏幕清空；BackLight = 1 是将 LED 背光灯打开。

NUE-PSK3.1 的程序中多处对 LCD 进行操作，既有字符显示也有图形显示。对 LCD 操作的最基本的函数是 Write_Display_Data (uchar data, uchar rev)，该函数将 Data 显示到当前位置。

此外 NUE-PSK3.1 还通过 RD1 引脚输出能发出报警声音的蜂鸣器，其电路如图 7−12 所示。

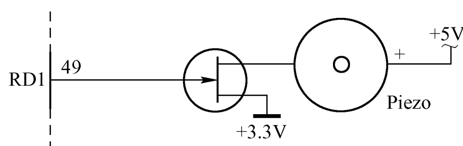


图 7−12　蜂鸣器接口电路图

7.2.4　数/模与模/数接口

NUE-PSK3.1 调制解调器的模拟接口采用 8 芯航空插头与单边带电台的音频接口相连。音频模拟信号的输入由 8 芯航空插头的第 7 脚输入，经过 MCP6S21 可编程增益放大器输入到 dsPIC33FJ128MC706 的 AN0 输入端；经过调制后的输出信号通过 SPI 传送给 12 位 D/A 转换器 MCP4922，最后由 8 芯航空插头的第 4 脚输出。

7.2.5 模/数接口

输入音频模拟信号的模/数转换由 dsPIC33FJ128MC706 内部的 ADC 模块实现。由于输入信号的幅度不一定合适，在 A/D 转换之前需要对幅度进行调整。NUE-PSK3.1 使用了一片通过 SPI 进行可编程的增益放大器 MCP6S21 对输入信号进行整形。模拟信号输入接口电路如图 7-13 所示。

MCP6S21 是 Microchip 公司针对 A/D 转换驱动设计的一款 SPI 接口的可编程低增益放大器（PGA），其中引脚 SCK 为 SPI 时钟输入、SDI 为 SPI 串行输入、CS 为 SPI 片选，Vref 为外部参考电压引脚，这里的 Vref 取 3.3V 的 1/2，即 1.65V 作为参考电压。Vref 的产生是由 MCP601 运放实现的，用于确保电压的稳定可靠。

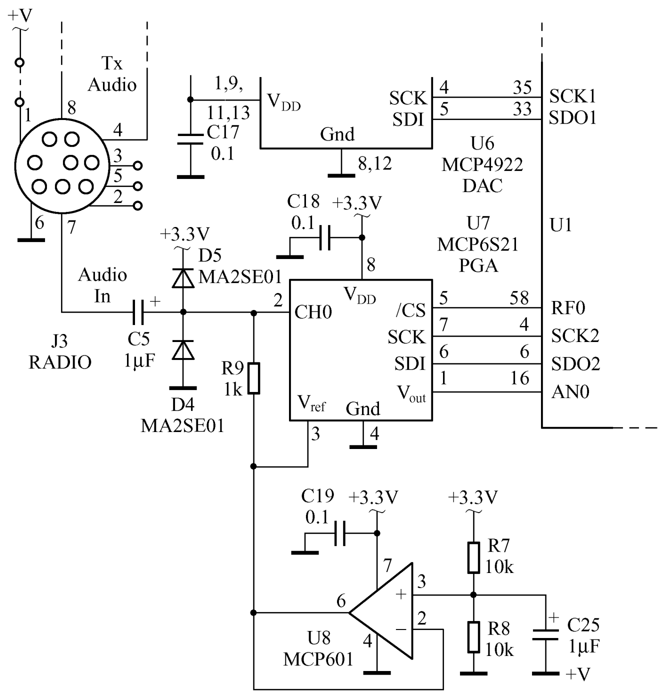


图 7-13　模拟信号输入接口电路图

当 dsPIC33FJ128MC706 将片选 CS 置为低时，启动与 MCP6S21 的通信。每个 SI 字（双字节长）的第一个字节是指令字节，进入指令寄存器，第二个字节为数据字节，指向指令字节的目标单元。其信号时序如图 7-14 所示。

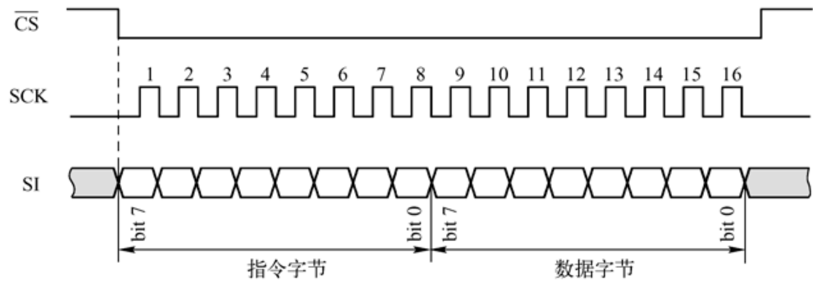


图 7-14　MCP6S21 SPI 接口时序图

MCP6S21 内部有三个 8 位寄存器：指令寄存器、增益寄存器和通道寄存器。其中增益寄存器的低 3 位表示+1~+32 的增益。

对 SPI 的初始化是由源文件 NUE_PSK_main_1.c 中的 Init_SPI ()函数完成的，该函数在 NUE_PSK_init_SPI.c 文件中定义：

```
void  init_SPI ( void ) {

/*=======================================================================*/

    MCP6S21_CS1_tris     =    0;
    Nop()
    MCP4922_CS1_tris     =    0;
    Nop()
    MCP6S21_CS1          =    1;
    Nop()
    MCP4922_CS1          =    1;
    Nop()

/*=======================================================================*/
/*----- SPI1CON1 Bits ----------------------------------------------------*/

    SPI1CON1bits.PPRE    =    2;
    Nop()
    SPI1CON1bits.SPRE    =    7;
    Nop()
    SPI1CON1bits.MSTEN       =    1;
    Nop()
    SPI1CON1bits.CKP     =    0;
    Nop()
    SPI1CON1bits.SSEN    =    0;
    Nop()
    SPI1CON1bits.CKE     =    1;
    Nop()
    SPI1CON1bits.SMP     =    0;
    Nop()
    SPI1CON1bits.MODE16= 1;
    Nop()
    SPI1CON1bits.DISSDO      =    0;
    Nop()
    SPI1CON1bits.DISSCK      =    0;
    Nop()

/*=======================================================================*/
/*----- SPI1CON2 Bits ----------------------------------------------------*/

    SPI1CON2bits.FRMDLY      =    1;
```

```
        Nop()
        SPI1CON2bits.FRMPOL     =       1;
        Nop()
        SPI1CON2bits.SPIFSD=    0;
        Nop()
        SPI1CON2bits.FRMEN      =       0;
        Nop()

/*========================================================================*/
/*----- SPI1STAT Bits --------------------------------------------------*/

        SPI1STATbits.SPISIDL    =       0;
        Nop()
        SPI1STATbits.SPIEN      =       1;
        Nop()
        SPI1STATbits.SPIROV     =       0;
        Nop()

/*========================================================================*/
/*----- SPI2CON1 Bits --------------------------------------------------*/

        SPI2CON1bits.PPRE   =   2;
        Nop()
        SPI2CON1bits.SPRE   =   7;
        Nop()
        SPI2CON1bits.MSTEN      =       1;
        Nop()
        SPI2CON1bits.CKP    =   0;
        Nop()
        SPI2CON1bits.SSEN   =   0;
        Nop()
        SPI2CON1bits.CKE    =   1;
        Nop()
        SPI2CON1bits.SMP    =   0;
        Nop()
        SPI2CON1bits.MODE16=    1;
        Nop()
        SPI2CON1bits.DISSDO     =       0;
        Nop()
        SPI2CON1bits.DISSCK     =       0;
        Nop()

/*========================================================================*/
/*----- SPI2CON2 Bits --------------------------------------------------*/

        SPI2CON2bits.FRMDLY     =       1;
```

```
        Nop()
        SPI2CON2bits.FRMPOL     =     1;
        Nop()
        SPI2CON2bits.SPIFSD=    0;
        Nop()
        SPI2CON2bits.FRMEN      =     0;
        Nop()

    /*=======================================================================*/
    /*---- SPI2STAT Bits -----------------------------------------------------*/

        SPI2STATbits.SPISIDL    =     0;
        Nop()
        SPI2STATbits.SPIEN      =     1;
        Nop()
        SPI2STATbits.SPIROV     =     0;
        Nop()

        }
```

dsPIC33FJ128MC706 通过 SPI2 与 MCP6S21 相连。其中 SPI2CON1bits.MODE16=1 表示数据宽度为 16 位；SPI2CON1bits.MSTEN=1 表示 dsPIC33FJ128MC706 工作在主模式。

对 MCP6S21 操作的代码由 NUE_PSK_Subroutines_2C.c 文件中的函数 Set_PGA1_Gain (int gain_index)定义：

```
    void  Set_PGA1_Gain  (      int gain_index )   {

        int     temp;
        int gain;

        switch (gain_index) {
            case 0:
                gain = x1;
                 break;
            case 1:
                gain = x5;
                break;
            case 2:
                gain = x16;
                break;
            case 3:
                gain = x32;
                break;
            default:
                break;
        }
```

```
    MCP6S21_CS1   =     0;

    SPI2BUF         =     (    ( gain & 0x0007 ) | 0x4000    );
    while ( !SPI2STATbits.SPIRBF );
    temp         =      SPI2BUF;
        MCP6S21_CS1   =     1;

    }
```

在对 MCP6S21 操作之前先将其片选信号拉低，然后将命令字写入寄存器 SPI2BUF 中，这里选中的是增益寄存器，增益值由该函数的输入参数 gain_index 决定，等待写入成功后将片选引脚拉高。

对 ADC 的初始化由源文件 NUE_PSK_main_1.c 中的 Init_ADC ()函数完成，该函数在 NUE_PSK_init_ADC.c 文件中定义：

```
void  init_ADC ( void ) {

/*----- AD1CON1 Bits ------------------------------------------------------*/

    AD1CON1bits.DONE  =     0;
    Nop()
    AD1CON1bits.SAMP  =     0;
    Nop()
    AD1CON1bits.ASAM  =     1;
    Nop()
    AD1CON1bits.SIMSAM     =     0;
    Nop()
    AD1CON1bits.SSRC   =     7;
    Nop()
    AD1CON1bits.FORM  =     3;
    Nop()
    AD1CON1bits.AD12B =     1;
    Nop()
    AD1CON1bits.ADDMABM =     1;
    Nop()
    AD1CON1bits.ADSIDL     =     0;
    Nop()
    AD1CON1bits.ADON  =     1;

/*=====================================================================*/
/*----- AD1CON2 Bits ------------------------------------------------------*/

    AD1CON2bits.ALTS   =     0;
    Nop()
    AD1CON2bits.BUFM  =     0;
```

```
Nop()
AD1CON2bits.SMPI   =    0;
Nop()
AD1CON2bits.BUFS   =    0;
Nop()
AD1CON2bits.CHPS   =    0;
Nop()
AD1CON2bits.CSCNA=      0;
Nop()
AD1CON2bits.VCFG   =    0;

/*=========================================================================*/
/*----- AD1CON3 Bits -----------------------------------------------------*/

AD1CON3bits.ADCS   =    5;
Nop()
AD1CON3bits.SAMC   =    1;
Nop()
AD1CON3bits.ADRC   =    0;

/*=========================================================================*/
/*----- AD1CHS123 Bits ---------------------------------------------------*/

AD1CHS123bits.CH123SA   =    0;
Nop()
AD1CHS123bits.CH123NA   =    0;
Nop()
AD1CHS123bits.CH123SB   =    0;
Nop()
AD1CHS123bits.CH123NB   =    0;

/*=========================================================================*/
/*----- AD1CHS0 Bits -----------------------------------------------------*/

AD1CHS0bits.CH0SA =     0;
Nop()
AD1CHS0bits.CH0NA =     0;
Nop()
AD1CHS0bits.CH0SB =     0;
Nop()
AD1CHS0bits.CH0NB =     0;

/*=========================================================================*/
/*----- AD1PCFGL Bits ----------------------------------------------------*/

AD1PCFGLbits.PCFG0      =    0;
Nop()
```

```
    AD1PCFGLbits.PCFG1     =   1;
    Nop()
    AD1PCFGLbits.PCFG2     =   1;
    Nop()
    AD1PCFGLbits.PCFG3     =   1;
    Nop()
    AD1PCFGLbits.PCFG4     =   1;
    Nop()
    AD1PCFGLbits.PCFG5     =   1;
    Nop()
    AD1PCFGLbits.PCFG6     =   1;
    Nop()
    AD1PCFGLbits.PCFG7     =   1;
    Nop()
    AD1PCFGLbits.PCFG8     =   1;
    Nop()
    AD1PCFGLbits.PCFG9     =   1;
    Nop()
    AD1PCFGLbits.PCFG10    =   1;
    Nop()
    AD1PCFGLbits.PCFG11    =   1;
    Nop()
    AD1PCFGLbits.PCFG12    =   1;
    Nop()
    AD1PCFGLbits.PCFG13    =   1;
    Nop()
    AD1PCFGLbits.PCFG14    =   1;
    Nop()
    AD1PCFGLbits.PCFG15    =   1;

/*==================================================================*/
/*----- AD1CSSL Bits ---------------------------------------------------*/

    AD1CSSLbits.CSS0   =   1;
    Nop()
    AD1CSSLbits.CSS1   =   0;
    Nop()
    AD1CSSLbits.CSS2   =   0;
    Nop()
    AD1CSSLbits.CSS3   =   0;
    Nop()
    AD1CSSLbits.CSS4   =   0;
    Nop()
    AD1CSSLbits.CSS5   =   0;
    Nop()
    AD1CSSLbits.CSS6   =   0;
    Nop()
    AD1CSSLbits.CSS7   =   0;
```

```
        Nop()
        AD1CSSLbits.CSS8   =   0;
        Nop()
        AD1CSSLbits.CSS9   =   0;
        Nop()
        AD1CSSLbits.CSS10  =   0;
        Nop()
        AD1CSSLbits.CSS11  =   0;
        Nop()
        AD1CSSLbits.CSS12  =   0;
        Nop()
        AD1CSSLbits.CSS14  =   0;
        Nop()
        AD1CSSLbits.CSS15  =   0;

    /*==========================================================================*/
    /*----- AD1CON4 Bits -----------------------------------------------------*/

        AD1CON4bits.DMABL      =   0;

    /*==========================================================================*/
    ……
    }
```

这里初始化了两个 ADC，前一个为模拟音频输入的 ADC 控制器。

其中 AD1CON1bits.AD12B=1 表示采用 12 位 1 通道 ADC。

A/D 采集到的数据将进行解调处理，其代码为 NUE_PSK_Subroutines_2C.c 文件中的函数 Input_AD1 (void)：

```
    int      Input_AD1 (void)     {

             while ( !AD1CON1bits.DONE );
             AD1CON1bits.DONE  =   0;
             return    ADC1BUF0;

        }
```

等待 A/D 转换结束后直接将转换后的结果 ADC1BUF0 输出。

7.2.6　数/模接口

NUE-PSK3.1 将调制后的数字信号通过 SPI 输入到 12 位 DAC 器件 MCP4922 中进行数/模转换。MCP4922 是 Microchip 公司推出的高精度、低噪声、工业级的数/模转换器，这里用到的 MCP4922 有两路输出，实际电路只用到了一个。MCP4922 的参考电压取该芯片的工作电压 3.3V。模拟输出接口电路原理如图 7−15 所示。

dsPIC33FJ128MC706 通过 SPI1 与 MCP4922 相连，对于 MCP4922 而言这是一个只写的接口。当 CS 片选引脚为低时，dsPIC33FJ128MC706 向 MCP4922 写入 16 位的命令寄存器数据，其定义如图 7−16 所示。

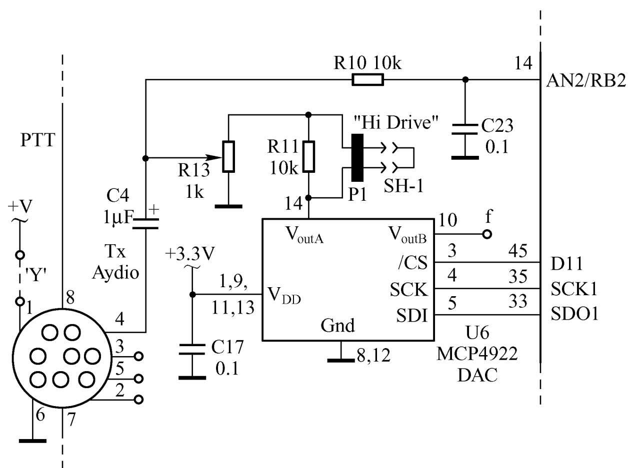


图 7-15　模拟输出接口原理图

Upper Half:

W-x	W-x	W-x	W-0	W-x	W-x	W-x	W-x
$\overline{A}$/B	BUF	$\overline{GA}$	$\overline{SHDN}$	D11	D10	D9	D8
bit 15							bit 8

Lower Half:

W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
D7	D6	D5	D4	D3	D2	D1	D0
bit 7							bit0

图 7-16　MCP4922 命令寄存器定义

其中 bit15 选择 A 或 B 通道输出；bit13 为输出增益控制，当该位为 0 时增益为 2X，为 1 时增益为 1X；bit11～bit0 为 12 位 DAC 数据。

对 MCP4922 操作的代码在 NUE_PSK_Subroutines_2C.c 文件中，包括函数 Output_DAC_1A (int data)和 Output_DAC_1B (int data)，其中 data 为经过调整后的数字信息。

```
void  Output_DAC_1A ( int  data ) {

        int     temp;
        MCP4922_CS1   =     0;
        SPI1BUF    =     ( ( data & 0x0FFF ) | 0x3000 );
        while ( !SPI1STATbits.SPIRBF );
        temp =      SPI1BUF;
        MCP4922_CS1   =     1;

    }
```

对 MCP4922 写入数据前要将其片选信号置低：

```
MCP4922_CS1=0;
```

将处理后的数据直接送到 SPI1BUF 中，这里使用了 A 通道输出、增益为 1X：

```
SPI1BUF=( ( data & 0x0FFF ) | 0x3000 );
```

等待数据成功写入后将片选信号拉高。

7.2.7　I2C 外部存储接口

NUE-PSK3.1 调制解调器的配置信息需要保存在外部可读写存储器中。这里采用了 I2C 接口的 EEPROM 存储器 24AA256。该存储器为 256KB 的 CMOS 非易失存储器，包含三位译码地址，且具有写保护功能，内部 64 个字节为一页。24AA256 与 dsPIC33FJ128MC706 的接口原理如图 7−17 所示。

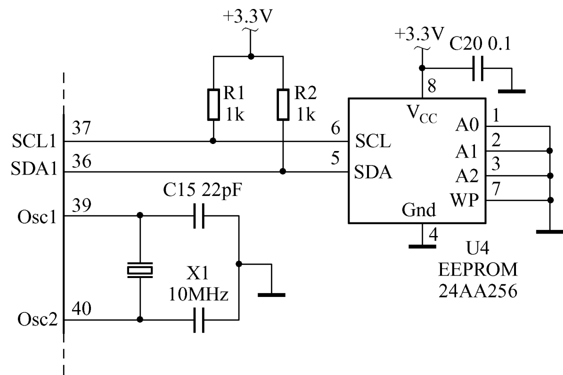


图 7−17　24AA256 接口原理图

该电路中 24AA256 的地址选取 000，硬件写保护失效。24AA256 为从设备，通过 I2C 接口与 dsPIC33FJ128MC706 相连。对 24AA256 的操作分为读、写两个部分。

读操作分为当前读、随机读和顺序读三种。当前读的时序如图 7−18 所示。

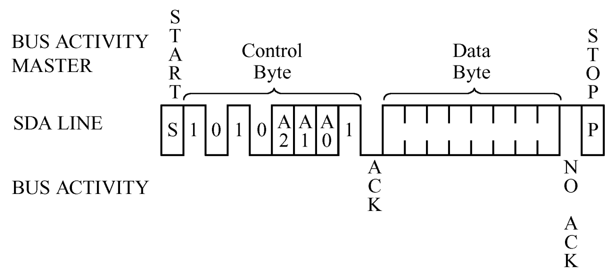


图 7−18　24AA256 当前读时序图

随机读需要先向 24AA256 发送两个地址字节，然后再读取数据。随机读的时序如图 7−19 所示。

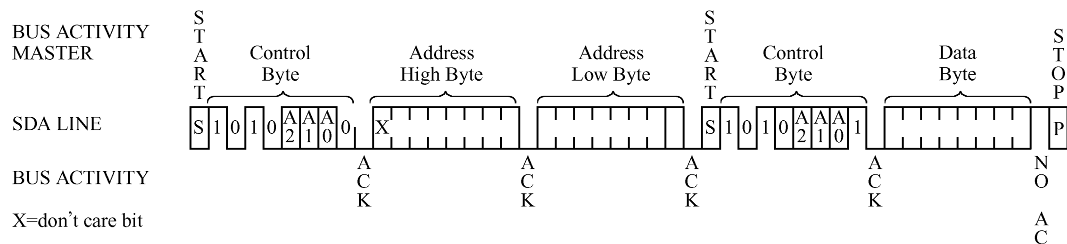


图 7−19　24AA256 随机读时序图

顺序读是在随机读的基础上连续读出 N 个字节。

写操作分为单字节写和整页写两种。单字节写的时序如图 7-20 所示。

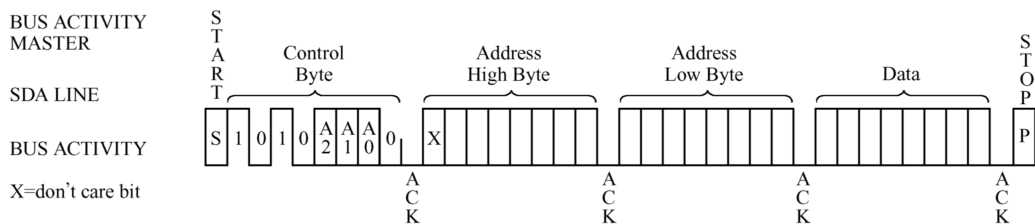


图 7-20　24AA256 单字节写时序图

而整页写是在单字节写的基础上连续写入 64 个字节（见图 7-21）。

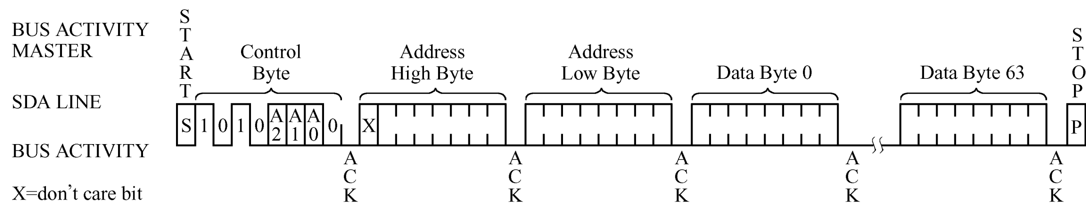


图 7-21　24AA256 整页写时序图

dsPIC33FJ128MC706 对 24AA256 的初始化代码由 NUE_PSK_main_1.c 中的 Init_I2C1 () 函数完成，该函数在 NUE_PSK_I2C.c 文件中定义，具体代码如下：

```
void init_I2C1    ( void )     {

        I2C1RCV                    =     0;
        I2C1TRN                    =     0;
        I2C1ADD                    =     0;
        I2C1MSK                    =     0;

        I2C1BRG                    =     363;

        I2C1CONbits.SEN       =     0;
        Nop()
        I2C1CONbits.RSEN      =     0;
        Nop()
        I2C1CONbits.PEN       =     0;
        Nop()
        I2C1CONbits.RCEN      =     0;
        Nop()
        I2C1CONbits.ACKEN     =     0;
        Nop()
        I2C1CONbits.ACKDT     =     0;
        Nop()
        I2C1CONbits.STREN     =     0;
        Nop()
        I2C1CONbits.GCEN      =     0;
```

```
Nop()
I2C1CONbits.SMEN        =   0;
Nop()
I2C1CONbits.DISSLW      =   0;
Nop()
I2C1CONbits.A10M        =   0;
Nop()
I2C1CONbits.IPMIEN      =   0;
Nop()
I2C1CONbits.SCLREL      =   0;
Nop()
I2C1CONbits.I2CSIDL     =   0;
Nop()
I2C1CONbits.I2CEN       =   1;

I2C1STATbits.TBF        =   0;
Nop()
I2C1STATbits.RBF        =   0;
Nop()
I2C1STATbits.R_W        =   0;
Nop()
I2C1STATbits.S          =   0;
Nop()
I2C1STATbits.P          =   0;
Nop()
I2C1STATbits.D_A        =   0;
Nop()
I2C1STATbits.I2COV      =   0;
Nop()
I2C1STATbits.IWCOL      =   0;
Nop()
I2C1STATbits.ADD10      =   0;
Nop()
I2C1STATbits.GCSTAT     =   0;
Nop()
I2C1STATbits.BCL        =   0;
Nop()
I2C1STATbits.TRSTAT     =   0;
Nop()
I2C1STATbits.ACKSTAT    =   0;

}
```

对 24AA256 的操作由 I2C1_Byte_Write (int Write_Address, char Write_Data) 和 I2C1_Byte_Read (int Read_Address) 函数完成。这两个函数也在 NUE_PSK_I2C.c 文件中定义，I2C1_Byte_Write (int Write_Address, char Write_Data) 函数用于从 M24AA256 中读数据，具体代码如下：

```
void  I2C1_Byte_Write( int Write_Address, char Write_Data )    {

      I2C1CONbits.SEN       =      1;
      while ( I2C1CON & 0b11111 );

      I2C1TRN   =      EEPROM_Write_Address;
      while ( I2C1STATbits.TRSTAT );

      I2C1TRN   =      Write_Address>>8;
      while ( I2C1STATbits.TRSTAT );

      I2C1TRN   =      Write_Address;
      while ( I2C1STATbits.TRSTAT );

      I2C1TRN   =      Write_Data;
      while ( I2C1STATbits.TRSTAT );

      I2C1CONbits.PEN       =      1;
      while ( I2C1CONbits.PEN );

      Delay_5ms();
      Delay_5ms();
      }
```

I2C1_Byte_Read (int Read_Address) 函数用于实现向 24AA256 写入数据的功能，具体代码如下：

该函数实现向 24AA256 写入数据的功能。

```
char  I2C1_Byte_Read ( int Read_Address )    {

      char  Read_Data;

      I2C1CONbits.SEN       =      1;
      while ( I2C1CONbits.SEN );

      I2C1TRN   =      EEPROM_Write_Address;
      while ( I2C1STATbits.TRSTAT );

      I2C1TRN   =      Read_Address>>8;
      while ( I2C1STATbits.TRSTAT );

      I2C1TRN   =      Read_Address;
      while ( I2C1STATbits.TRSTAT );

      I2C1CONbits.RSEN      =      1;
      while ( I2C1CONbits.RSEN );

      I2C1TRN   =      EEPROM_Read_Address;
      while ( I2C1STATbits.TRSTAT );

      while ( I2C1STATbits.ACKSTAT );

      I2C1CONbits.RCEN      =      1;
```

```
while ( I2C1CONbits.RCEN );

Read_Data =     I2C1RCV;

I2C1CONbits.ACKDT =     1;
I2C1CONbits.ACKEN =     1;
while ( I2C1CONbits.ACKEN );

I2C1CONbits.PEN =   1;
while ( I2C1CONbits.PEN );

return  Read_Data;

}
```

7.3 软件程序概况

软件部分共有程序中 19 个 C 文件，分别是：

- NUE_PSK_main_1.c：主程序。
- PSKMod.c：调制程序。
- PskDet.c：解调程序。
- uart.c：串口处理程序。
- NUE_PSK_TIM1_isr.c：定时器中断处理程序。
- NUE_PSK_Subroutines_2C.c：子程序。
- NUE_PSK_Sine_table.c：正弦表。
- NUE_PSK_QED.c：QED 编码接口初始化程序。
- NUE_PSK_LCD_single_1.c：LCD 处理程序。
- NUE_PSK_Keyboard_Functions.c：键盘处理的子程序。
- NUE_PSK_Keyboard.c：键盘处理程序。
- NUE_PSK_init_TIM1.c：定时器初始化程序。
- NUE_PSK_init_SPI.c：SPI 初始化程序。
- NUE_PSK_init_clock.c：时钟初始化程序。
- NUE_PSK_init_ADC.c：AD 初始化程序。
- NUE_PSK_I2C.c：I2C 处理程序。
- NUE_PSK_Globals.c：一些全局变量。
- NUE_PSK_FIR.c：FIR 滤波器初始化处理。
- NUE_PSK_Coefficients.c：滤波器的抽头系数。

其中，NUE_PSK_main_1.c 是软件的主程序，它分别调用其他 18 个文件完成 PSK 信号的调制解调、键盘输入、LCD 显示等工作。下面分别按照发送端软件和接收端软件来进行介绍。

7.4 发送端软件

发送端软件的结构框图如图 7-22 所示。发送端键盘输入的字符首先送入字符 FIFO。

FIFO 的输出送入可变长信源编码，得到 3~15bit 的输出，编码的长度是根据字符出现的概率来确定的，经常出现的字符编码的长度比较短。编码的输出送入 BPSK/QPSK 串并转换和差分编码，得到差分编码的数据。然后再送入波形成型滤波和调制，得到调制后的信号。调制信号送入 D/A 转换器，得到模拟信号，再送入发射机即可。

数据发送的时候首先发送 32bit 的 1、0 交替的前导符号，然后发送信息，结束的时候再发送 32bit 的 1 的结束符号。前导符号在接收端用来进行载波同步。

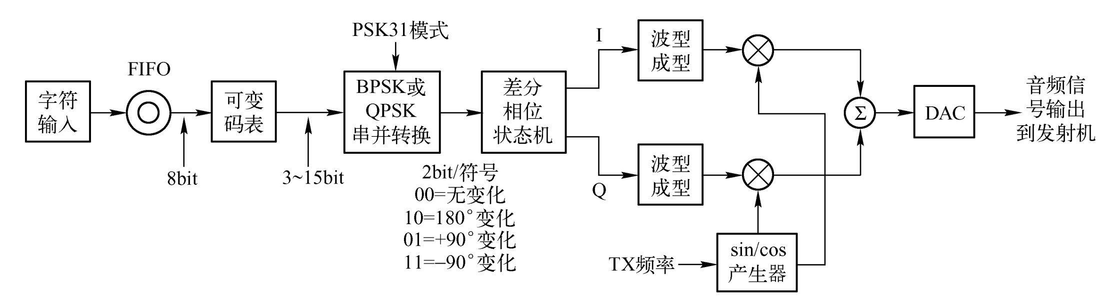


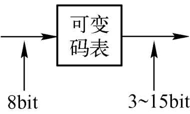
图 7−22　发送端软件的结构框图

7.4.1　可变长编码

（1）功能

完成 8bit 的 ASCII 码到 3～15bit 的可变长编码的转换。

（2）功能框图



输入：8bit 的 ASCII 码

输出：可变长 3～15bit 编码输出

（3）数据结构

数组 VARICODE_TABLE[256]，是一个 256 个元素的数组。

（4）程序说明

文件 PskMod.c 的第 499 行：

```
m_TxShiftReg = VARICODE_TABLE[ ch&0xFF ];
```

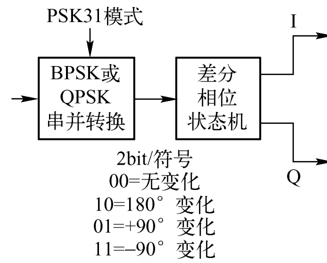
其中，变量 ch 是从键盘输入得到的 8bit ASCII 字符。程序通过 ch&0xFF 对数组 VARICODE_TABLE 进行查表，得到的 3～15bit 信息送入变量 m_TxShiftReg。

7.4.2　BPSK/QPSK 串/并转换和差分编码

（1）功能

对 BPSK 完成差分编码，对 QPSK 完成串/并转换和差分编码。

（2）功能框图



输入：可变长编码得到的比特流信息。

输出：编码得到相位变化信息，送入成型滤波和调制。

（3）程序说明

文件 PskMod.c 的第 466 行。

```
/*================================*/
/* 每个比特调用一次，得到要传输的符号*/
/*================================*/
char GetNextBPSKSymbol(void)
{
 char symb;
 char ch;
 symb = m_Lastsymb;

 if( m_TxShiftReg == 0 )
 {
      if( m_AddEndingZero )        /* 判断是否到一个字符的结束*/
      {
            symb = SYM_P180;            /* 以 0 结尾 */
            m_AddEndingZero = FALSE;
      }
      else
      {

            ch = GetChar();                    /*得到下一个要发送的字符 */
            switch( ch ) {
                  case TXON_CODE:
                        symb = SYM_ON;
                        break;
                  case TXTOG_CODE:
                        symb = SYM_P180;
                        break;
                  case TXOFF_CODE:
                        symb = SYM_OFF;
                        break;
                  default: /*得到下一个要发送的可变长字符 */
```

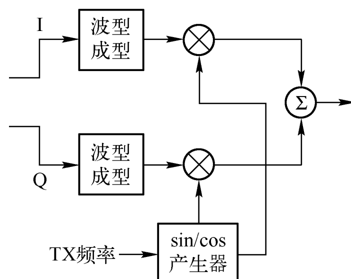
```
                    m_TxShiftReg = VARICODE_TABLE[ ch&0xFF ];
                    symb = SYM_P180;    /*开始是一个 0 */
                    break;
            }
        }
    }
    else                /* 判断不是字符的结束，开始发送下一个比特 */
    {
        if( m_TxShiftReg&0x8000 )
            symb = SYM_NOCHANGE; //差分编码：比特为 1，波形不变化
        else
            symb = SYM_P180;          //差分编码：比特为 0，波形变化 180°
        m_TxShiftReg = m_TxShiftReg<<1;     /*得到下一个比特 */
        if( m_TxShiftReg == 0 )             /* 一个字符是否结束 */
            m_AddEndingZero = TRUE;         /* 下一个要多发送一个 0*/
    }
    m_Lastsymb = symb;
    return symb;
}
```

7.4.3 成型滤波和调制

（1）功能

完成对 I/Q 数据的成型滤波和调制功能。

（2）功能框图



输入：串/并转换和差分编码得到的 I/Q 数据。

输出：送入到 DA，得到模拟信号，再送入发射机。

（3）程序说明

文件 PskMod.c 的第 260 行。

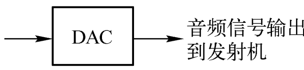
```
//得到成型滤波的结果
S1 = PSKEnv (Itbl_num,m_Ramp);
S2 = PSKEnv (Qtbl_num,m_Ramp);
//把成型滤波的结果和载波相乘，得到调制的结果
Product1 = (int)(m_RMSConstant * S1 * TX_I);
Product2 = (int)(m_RMSConstant * S2 * TX_Q);
```

7.4.4　数/模转换

（1）功能

把调制的数字域波形转换为模拟波形。

（2）功能框图



输入：调制器的输出。

输出：送入到发射机。

（3）程序说明及注释

文件 NUE_PSK_Subroutines_2C.c 中的第 170 行。

```
void  Output_DAC_1A ( int  data )
{
        int     temp;
        MCP4922_CS1   =      0;                              //打开 DA 器件 MCP4922
        SPI1BUF    =      ( ( data & 0x0FFF ) | 0x3000 );    //把数据通过 SPI 送入 DA 器件
        while ( !SPI1STATbits.SPIRBF );                      //监测数据是否成功送入 DA 器件
              temp =      SPI1BUF;
        MCP4922_CS1   =      1;                              //关闭 DA 器件
    }
```

7.5　接收端软件

接收端软件的结构框图如图 7-23 所示。接收机得到的模拟信号首先送入到 AD 器件，得到采样率为 8000Hz 的采样数据。采样数据一路进行 FFT 计算，得到信号的功率谱，送入到 LCD 进行显示，方便观察信道的状态和调谐信号；另外一路送入到解调器和本地载波相乘，进行解调，得到基带 I/Q 信号。基带 I/Q 信号经过两级 4∶1 的 35 抽头的抽取滤波器，得到采样率为 500Hz 的 I/Q 数据，这样做可以降低系统的计算量的要求，也可以提高系统的抗噪声性能。抽取滤波器的输出 I/Q 数据一路送入到 65 抽头的匹配滤波器，进行滤波，最大限度地去除噪声，提高采样时刻的信噪比；另外一路送入 65 抽头的频率滤波器，以进行载波同步（AFC：自动频率控制）。频率滤波器的结果送入到 AGC 电路，进行自动增益控制，然后送入到载波同步电路进行相差和频差计算，反馈到 NCO（数字控制振荡器）来控制本地载波的相位，从而实现相干解调。匹配滤波器的输出经过增益控制后一路送入到符号同步电路，经过计算得到最佳的抽样判决时刻；另外一路送入到抽样判决电路，在符号同步电路的控制下进行抽样和判决工作。抽样判决的信号送入到差分译码和 viterbi 译码电路，经过计算得到原始发送的比特信息（这里由于噪声的原因，可能会出现误码）。然后译码的信息送入到信源译码电路，进行可变长译码工作，得到 8bit 的 ASCII 字符。另外差分译码的结果送入到静噪控制和信号质量计算电路，计算信号质量，从而决定是否进行静噪控制。

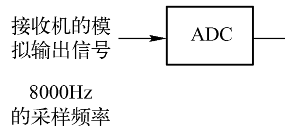
图 7-23　接收端软件的结构框图

7.5.1　模/数转换

（1）功能

把模拟波形转换为数字域波形，采样速率为 8000Hz。

（2）功能框图



输入：接收机的模拟输出信号。

输出：送入到抽取滤波器。

（3）数据结构

数组 pADC_Buffer 为一个长度为 2×1024 的输入缓存，分为两个长度为 1024 的缓存，当采样满 1024 个采样点后，开始对这一块长 1024 的数据进行后续信号处理。

（4）程序说明及注释

文件 NUE_PSK_TIM1_isr.c 中的第 38 行：

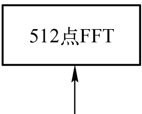
```
pADC_Buffer[Intrpt_Count] = Input_AD1();
```

7.5.2　512 点 FFT

（1）功能

计算采样得到的信号的功率谱，并在 LCD 上进行显示，以方便进行信号的调谐。程序中调用了 dsp 库分别实现“in-place”复数 FFT 变换、比特反转、功率谱计算、简单 IIR 滤波和 dB 格式转换等功能。

（2）功能框图



输入：模/数转换后采样的 512 一组的信号数据。

输出：送入到 LCD 中进行显示。

（3）数据结构

数组 unsigned int FFT_Output_Buffer [512]是 FFT 缓存，可以存储 512 个 AD 采样的数据，在程序中对这 512 个数据进行 FFT 变换，再经过计算得到信号的功率谱。

（4）程序说明

文件 PskDet.c 中的第 1183 行：

```
/*  对长度为 512 的 FFT 缓存进行 FFT 变换*/
void Proc_FFT ( void ) {
    int     i;
```

```
        extern Flag_type Flag;
        extern unsigned int FFT_Output_Buffer [];
        extern unsigned int FLOB [];
        extern const fractcomplex    twiddleFactors[FFT_BLOCK_LENGTH/2]
        __attribute__ ((space(auto_psv), aligned (FFT_BLOCK_LENGTH*2)));
    /*调用函数，进行"in-place"  复数 FFT 变换*/
        FFTComplexIP (LOG2_BLOCK_LENGTH, &FFT_buffer[0],
            (fractcomplex *)__builtin_psvoffset(&twiddleFactors[0]),
            (int)  __builtin_psvpage(&twiddleFactors[0]));

    /*进行 FFT 中特有的比特反转操作*/
        BitReverseComplex (LOG2_BLOCK_LENGTH, &FFT_buffer[0]);

    /*    计算复数 FFT 的幅度的平方，从而得到一个实数输出*/
        SquareMagnitudeCplx(FFT_BLOCK_LENGTH,&FFT_buffer[0], FFT_buffer[0].real);

    /*   对输出信号做一个简单的 IIR 滤波  */
        for (i=0; i<FFT_BLOCK_LENGTH/2; i++) {
        FFT_Output_Buffer [i] = ((3*FFT_Output_Buffer [i] +
            (unsigned int)FFT_buffer[FFT_BLOCK_LENGTH/2 - i].real)>>2);
        }
    /*    把结果转换到 dB 格式        */
        for (i=0; i<FFT_BLOCK_LENGTH/4; i++) {
            FLOB [i] = (int) 6.*log10((float) FFT_Output_Buffer [i+32] + 1.);
        }
        FLOB [0] = 0;

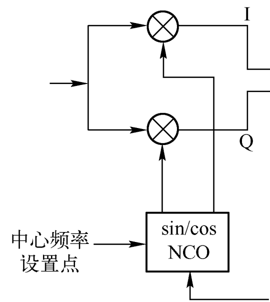
    /*    在 LCD 上显示频谱计算的结果  */
        Display_FFT ();
        if (m_PSKmode == RTTY_MODE)
    {
            Display_RTTYCursor();
            Display_RTTY_Info();
        }
        /*  清空 FFT 缓存  */
        for (i=0; i<FFT_BLOCK_LENGTH; i++)
    {
        FFT_buffer[i].real = 0.;
        FFT_buffer[i].imag = 0.;
    }
}
```

7.5.3 解调

（1）功能

对数/模转换后的信号进行解调，把频带信号转换为基带信号。

（2）功能框图



输入：数/模转换后得到的信号数据。

输出：送入后续的抽取滤波器匹配滤波和频率滤波模块。

（3）数据结构

程序中，Sine_lookup()函数通过查数组 Sine_table[1024] 得到本地载波的抽样值。数组 Sine_table[1024]是 1/4 周期的正弦表，程序中通过简单的处理得到其他 3/4 周期的正弦波数值。

（4）程序说明

文件 PskDet.c 中的第 400 行：

```
//通过查表法，得到正弦波
NCO_Q    =      ( Sine_lookup ( (NCO_phz >> 4 )    & 0xFFF ) ) ;
NCO_I    =      ( Sine_lookup ( ( ( NCO_phz >> 4 ) + 0x400 ) & 0xFFF ) ) ;

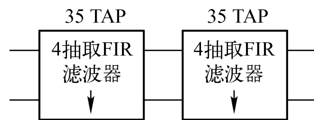
//把正弦波和接收信号相乘，进行解调
Product_I         = (long)      ADC_I * (long) NCO_I;
Product_Q  = (long)      ADC_I * (long) NCO_Q;
```

7.5.4 抽取滤波器

（1）功能

对解调后的 I/Q 数据进行抽取和滤波，结果送入信号后续匹配滤波和频率滤波模块。

（2）功能框图



输入：解调后的 I/Q 数据。

输出：送入后续匹配滤波和频率滤波模块。

（3）数据结构

程序中，Filter_I1/Q1 为抽取滤波器的结构，定义了抽取滤波器的抽头系数等参数；t Moe_35_sample_I_1 [0]为输入的采样信号；Moe_35_sample_I_2 [Index_2]为滤波输出的信号，抽取是按照 4∶1 来抽取的，即每 4 个抽样值，输出一个抽样信号。

（4）程序说明

文件 PskDet.c 中的第 421 行：

```
//调用 DSP 库中的抽取滤波函数 FIRDecimate
FIRDecimate (    1,
                 &Moe_35_sample_I_2 [ Index_2 ],
                 &Moe_35_sample_I_1 [ 0 ],
                 &Filter_I1,
                 4 );

FIRDecimate (    1,
                 &Moe_35_sample_Q_2 [ Index_2 ],
                 &Moe_35_sample_Q_1 [ 0 ],
                 &Filter_Q1,
                 4 );
```

7.5.5 比特匹配滤波器

（1）功能

对抽取后 I/Q 数据进行匹配滤波，结果送入信号抽样判决模块。

（2）功能框图



输入：抽取后的 I/Q 数据。

输出：送入信号抽样判决模块。

（3）数据结构

程序中，Filter_I4/Q4 为滤波器的结构，定义了抽头系数等参数；temp_I /Q 为输入的采样信号；FIROut_I_ freq /Q 为滤波输出的信号。

（4）程序说明

文件 PskDet.c 中的第 454 行：

```
FIR(1,&FIROut_I_freq, &temp_I, &Filter_I4);
FIR(1,&FIROut_Q_freq, &temp_Q, &Filter_Q4);
```

7.5.6 频率滤波器

（1）功能

对抽取后 I/Q 数据进行滤波，结果送入同步模块进行载波同步。

（2）功能框图



输入：抽取后的 I/Q 数据。

输出：送入同步模块进行载波同步。

（3）数据结构

程序中，Filter_I3/Q3 为滤波器的结构，定义了抽头系数等参数；FIR_Output_I /Q 为输入的采样信号；FIROut_I_bit /Q 为滤波输出的信号。

（4）程序说明

文件 PskDet.c 中的第 452 行。

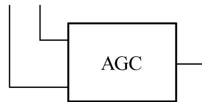
```
FIR(1,&FIROut_I_bit,    &FIR_Output_I, &Filter_I3);
FIR(1,&FIROut_Q_bit,    &FIR_Output_Q, &Filter_Q3);
```

7.5.7 AGC

（1）功能

利用滤波的结果进行自动增益计算，并对信号进行增益控制，从而达到稳定输出信号的目的。

（2）功能框图



输入：I/Q 数据。

输出：AGC 调节的系数。

（3）数据结构

程序中，m_AGCave 是一个 Double 型的变量，用来存储 AGC 的计算结果，并对信号进行增益控制。

（4）程序说明

文件 PskDet.c 中的第 815 行。

```
/* ////////////////////////////////////////////////////////////////    */
/*  自动增益控制计算                                                    */
/* ////////////////////////////////////////////////////////////////    */
void CalcAGC( struct Complex Samp)
{
        double mag;

        mag = sqrt(Samp.x*Samp.x + Samp.y*Samp.y);//计算得到信号幅度
        //若信号强，则采用 K=1/200 的 IIR 滤波器，否则采用 K=1/500 的 IIR 滤波器
        if( mag > m_AGCave )
                m_AGCave = (1.0-1.0/200.0)*m_AGCave + (1.0/200.0)*mag;
        else
                m_AGCave = (1.0-1.0/500.0)*m_AGCave + (1.0/500.0)*mag;
        //除非信号很小，否则对信号进行增益调节
        if( m_AGCave >= 1.0 )
        {
                m_BitSignal.x /= m_AGCave;
```

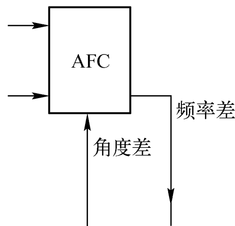
```
            m_BitSignal.y /= m_AGCave;
            m_FreqSignal.x /= m_AGCave;
            m_FreqSignal.y /= m_AGCave;
        }
    }
```

7.5.8 载波同步

（1）功能

利用 AGC 进行增益调整后的结果，计算本地载波的频差和相差，并调整 NCO（数控振荡器），从而达到相干解调的目的。

（2）功能框图



输入：AGC 增益调整的 I/Q 数据和相差。

输出：计算得到频差，并送入 NCO 进行频率调整。

（3）数据结构

程序中，m_AGCave 是一个 Double 型的变量，用来存储 AGC 的计算结果，并对信号进行增益控制。

（4）程序说明

文件 PskDet.c 中的第 740 行：

```
/* ///////////////////////////////////////////////////////////////////////////////    */
/*  计算载波和本地载波之间的频差，并控制 NCO   */
/* ///////////////////////////////////////////////////////////////////////////////    */
void CalcFreqError( struct Complex IQ )
{
    #define P_GN 0.001                    /*AFC  常数   */
    #define I_GN 1.5E-6
    #define P_CGN 0.0004
    #define I_CGN 3.0E-6
    #define WIDE_GN (1.0/.02)             /*把误差转换为 Hz 的常数   */
    #define WLP_K (200.0)
    extern double m_FferrAve;
    double freqerr;

    if(m_AFCmode == AFC_OFF)       //如果没有设置载波同步，则直接退出
    {
        m_FferrAve = 0.0;
```

```
		m_FperrAve = 0.0;
		m_FreqError = 0.0;
		return;
	}
	//计算得到瞬时频差
	freqerr = (IQ.x - m_z2.x) * m_z1.y - (IQ.y - m_z2.y) * m_z1.x;
	m_z2.x = m_z1.x;
	m_z2.y = m_z1.y;
	m_z1.x = IQ.x;
	m_z1.y = IQ.y;
	/*0.02 代表 1Hz*/
	if( freqerr > .30 )		/*对误差进行限幅，限制在正负 15Hz	*/
		freqerr = .30;
	if( freqerr < -.30 )
		freqerr = -.30;
	//对频差进行 IIR 滤波，得到一个稳定的平均频率误差，且结果可用 Hz 表示
	m_FferrAve = (1.0-1.0/WLP_K)*m_FferrAve + ((1.0*WIDE_GN)/WLP_K)*freqerr;

	if( m_AFCCaptureOn ) //如果已经捕获了
	{
		freqerr=m_FferrAve;
		if( (freqerr > 0.3) || (freqerr < -0.3 ) )//频差绝对值大于 0.3Hz，则调整 NCO 相位
			m_NCOphzinc = m_NCOphzinc + (freqerr*I_CGN);
		m_FreqError = freqerr*P_CGN;
	}
	else
	{
		if( (m_FferrAve*m_FperrAve)>0.0 )
			freqerr = m_FperrAve;
		else
			freqerr = 0.0;
		if( (freqerr > 0.3) || (freqerr < -0.3 ) ) //频差绝对值大于 0.3Hz，调整 NCO 相位
			m_NCOphzinc = m_NCOphzinc + (freqerr*I_GN);
		m_FreqError = freqerr*P_GN;
	}

	/*  对得到的频差进行限幅  */
	if( (m_NCOphzinc+m_FreqError) > m_AFCmax )
	{
		m_NCOphzinc = m_AFCmax;
		m_FreqError = 0.0;
	}
	else if( (m_NCOphzinc+m_FreqError) < m_AFCmin )
	{
		m_NCOphzinc = m_AFCmin;
		m_FreqError = 0.0;
```

```
        }

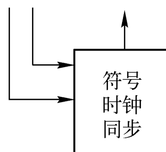
    }
```

7.5.9 位同步

（1）功能

利用 AGC 进行增益调整后的结果，计算最佳的采样位置，以得到最低的误码率。

（2）功能框图



输入：AGC 增益调整的 I/Q 数据。

输出：计算得到最佳采样位置，即位同步信息。

（3）程序说明

文件 PskDet.c 中的第 839 行：

```
/* ////////////////////////////////////////////////////////////////////       */
/*计算位同步的位置，主要是通过计算每一个抽  */
/*样时刻的脉冲的能量，然后进行平均，并挑选  */
/*能量最大的位置作为位同步的位置                */
/* ////////////////////////////////////////////////////////////////////       */
int SymbSync(struct Complex sample)
{
    int Trigger=FALSE;
    double max;
    double energy;
    int BitPos = m_BitPos;
    int i;

    if(BitPos<16)
    {
        energy = (sample.x*sample.x) + (sample.y*sample.y);//计算采样信号瞬时能量
        if( energy > 4.0)           /*对信号能量进行限幅，后续的 AGC 会进行调整*/
            energy = 1.0;
        //对瞬时能量进行一阶 IIR 滤波器
        m_SyncAve[BitPos] = (1.0-1.0/82.0)*m_SyncAve[BitPos] + (1.0/82.0)*energy;
        if( BitPos == m_PkPos )       /*  判断是否在最佳采样位置  */
        {
            Trigger = TRUE;
            m_SyncArray[m_PkPos] = (int)(900.0*m_SyncAve[m_PkPos]);
        }
        else
```

```
        {
                Trigger = FALSE;
                m_SyncArray[BitPos] = (int)(750.0*m_SyncAve[BitPos]);
        }
        //每 16 个采样点，循环一次
        if( BitPos == HALF_TBL[m_NewPkPos] )
                m_PkPos = m_NewPkPos;
        BitPos++;
}

m_BitPhasePos += (m_BitPhaseInc);
if( m_BitPhasePos >= Ts )
{
        m_BitPhasePos = fmod(m_BitPhasePos, Ts);    /*每 16 个采样点，循环一次*/
        if((BitPos==15) && (m_PkPos==15))
                Trigger = TRUE;
        BitPos = 0;
        max = -1e10;

        for(i=0; i<16; i++)        {//寻找最大能量的位置
                energy = m_SyncAve[i];
                if( energy > max ) {
                        m_NewPkPos = i;
                        max = energy;
                }
        }
        if(m_SQOpen) {
                if( m_PkPos == m_LastPkPos+1 )  /*计算时钟误差  */
                        m_ClkErrCounter++;
                else
                        if( m_PkPos == m_LastPkPos-1 )
                                m_ClkErrCounter--;
                if( m_ClkErrTimer++ > 313 )//每 10s 采样一次时钟的漂移
                {
                        m_ClkError = m_ClkErrCounter*200;     //采样时钟大约 200ppm 精度
                        m_ClkErrCounter = 0;
                        m_ClkErrTimer = 0;
                }
        }
        else
        {
                m_ClkError = 0;
                m_ClkErrCounter = 0;
                m_ClkErrTimer = 0;
        }
        m_LastPkPos = m_PkPos;
```

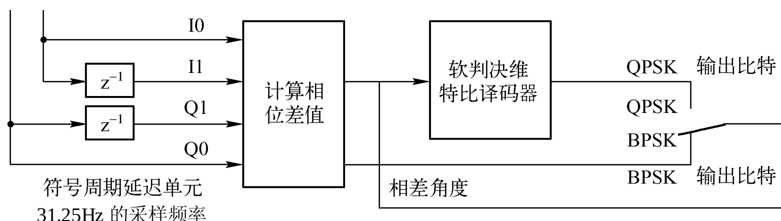
```
    }
    m_BitPos = BitPos;
    return Trigger;
}
```

7.5.10　差分译码

（1）功能

在 BPSK/QPSK 模式下，实现对信号的差分译码和软判决维特比译码，其中软判决译码在下面专门介绍。

（2）功能框图



输入：抽样判决后的 I/Q 数据。

输出：差分译码后数据送入到信源译码进行译码

（3）数据结构

m_I1/Q1 和 m_I2/Q2 为当前和前一个码元的 I/Q 数据，通过计算它们的相位差，即可得到结果。

（4）程序说明

文件 PskDet.c 中的第 970 行：

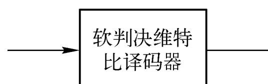
```
/*在 BPSK 模式下，通过两个码元的相位差来进行差分译码*/
vect.y = m_I1 * m_I0 +    m_Q1 * m_Q0;
bit = (int)(vect.y > 0.0);
```

7.5.11　软判决维特比译码

（1）功能

在 QPSK 模式下，实现对信号进行软判决维特比译码。

（2）功能框图



输入：QPSK 解调的输出。

输出：卷积码软判决维特比译码输出结果。

（3）程序说明

文件 PskDet.c 中的第 1135 行：

```
/*////////////////////////////////////////////////////////////////////    */
```

```
/*  软判决维特比译码器                              */
/* ////////////////////////////////////////////////////////////////////// */
int ViterbiDecode( double newangle)
{
        double pathdist[32];
        double min;
        int bitestimates[32];
        long ones;
        int i;
        const double* pAngleTbl;

        min = 1.0e37;                           /*  确保可以找到一个最小值*/
        if( newangle >= PI2/2 )                 /*  对不同的相位区间，采用两个不同的转换表格  */
                pAngleTbl = ANGLE_TBL2;         /*  用表格 2 */
        else
                pAngleTbl = ANGLE_TBL1;
        for(i = 0; i < 32; i++)                 /*  计算所有可能的距离，i 的 LSB 就是估计的比特  */
        {
                pathdist[i] = m_SurvivorStates[i / 2].Pathdistance +
                        fabs(newangle - pAngleTbl[ ConvolutionCodeTable[i] ]);
                if(pathdist[i] < min)
                        min = pathdist[i];
                /*  得到最新估计的比特  */
                bitestimates[i] = ((m_SurvivorStates[i / 2].BitEstimates) << 1) + (i & 1);
        }
        for(i = 0; i < 16; i++)                 /*  比较两个结束状态相同的路径的距离*/
                                                /*  把最小的路径存入 m_SurvivorStates[]. */
        {
                if(pathdist[i] < pathdist[16 + i])
                {
                        m_SurvivorStates[i].Pathdistance = pathdist[i] - min;
                        m_SurvivorStates[i].BitEstimates = bitestimates[i];
                }
                else
                {
                        m_SurvivorStates[i].Pathdistance = pathdist[16 + i] - min;
                        m_SurvivorStates[i].BitEstimates = bitestimates[16 + i];
                }
        }
        ones = 0;
        for(i = 0; i < 16; i++)                 /*比较估计出来的 0 多还是 1 多  */
                ones += (m_SurvivorStates[i].BitEstimates&(1L << 20));
        if( ones == (8L << 20 ) )//如果一样多，则输出一个随机值
                return ( rand() & 0x1000 );
        else                                    //否则输出一个比特
                return(ones > (8L << 20) );
```

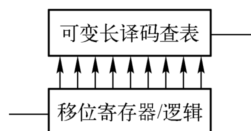
```
}
```

7.5.12 可变长信源译码

（1）功能

完成 3～15bit 可变长编码到 8bit 的 ASCII 码的转换。

（2）功能框图



输入：可变长 3～15bit 编码输出。

输出：8bit 的 ASCII 码。

（3）数据结构

数组 m_VaricodeDecTbl[2048]，是一个 2048 个元素的数组。

（4）程序说明

文件 PskMod.c 的第 499 行：

```
ch = m_VaricodeDecTbl[m_BitAcc];
```

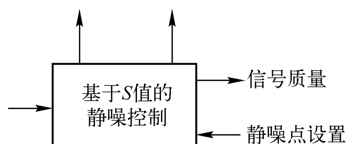
其中，变量 m_BitAcc 是解调（或维特比译码）得到的 3～15bit 的信息。程序通过 m_BitAcc 对数组 m_VaricodeDecTbl 进行查表，得到的结果送入变量 ch。

7.5.13 静噪控制和信号质量计算

（1）功能

利用解调得到的相位分布和设置的静噪电平，对信号的质量进行计算，从而提供信号质量、相差并进行静噪控制。

（2）功能框图



输入：解调得到的相位分布和设置的静噪电平。

输出：信号质量、相差和静噪控制。

（3）程序说明

文件 PskDet.c 中的第 1004 行：

```
/* ////////////////////////////////////////////////////////////////////         */
/*根据相差的统计特性，计算信号的质量                  */
/*相位和 0°、180°相位偏离越远，                    */
/*信号质量越差，这样就可以用来控制静噪了            */
```

```
/*如果有 20 个连续的 180° 相差，则打开静噪        */
/*如果有 20 个连续的 0° 相差，则关闭静噪          */
/* //////////////////////////////////////////////////////////////////////          */
void CalcQuality( double angle )
{
	#define ELIMIT 5
	#define PHZDERIVED_GN (1.0/.2)			/*把误差转换为以 Hz 为单位的增益常数*/
	double temp;
	double SqTimeK;

	SqTimeK = (double)m_SquelchSpeed;
	if( (m_PSKmode && ((angle >= PHZ_180_QMIN) && (angle <= PHZ_180_QMAX) ) ) ||
		(!m_PSKmode && ((angle >= PHZ_180_BMIN)&&(angle <= PHZ_180_BMAX))) )
	{	/*对 BPSK/QPSK 统计+/-45º 或者对 QPSK/L 统计  +/-180º 的情况*/
		if(m_PSKmode == QPSKL_MODE )
			temp = PI2/4.0 - angle;
		else
			temp = angle - PI2/4.0;
		m_QFreqError = temp;
		if( m_PSKmode ) /*在 QPSK 模式下  */
			temp = 280.0*fabs(temp);
		else
			temp = 150.0*fabs(temp);
		if( temp < m_DevAve)
			m_DevAve=	(1.0-1.0/SqTimeK)*m_DevAve + (1.0/SqTimeK)*temp;
		else
			m_DevAve=	(1.0-1.0/(SqTimeK*2.0))*m_DevAve + (1.0/(SqTimeK*2.0))*temp;
		if(m_OnCount > 20 )		/*  打开静噪  */
			m_DevAve = 100.0-75.0;
		else
			m_OnCount++;
		m_OffCount = 0;
		if( m_QFreqError >= 0.0 )
		{
			m_Pcnt++;
			m_Ncnt = 0;
		}
		else
		{
			m_Ncnt++;
			m_Pcnt = 0;
		}
		if( (m_Pcnt<ELIMIT) && (m_Ncnt<ELIMIT) )
			m_QFreqError = 0.0;
	}
	else
```

```
{
        if( (m_PSKmode && ((angle >= PHZ_0_QMIN) && (angle <= PHZ_0_QMAX) ) ) ||
                (!m_PSKmode && ((angle >= PHZ_0_BMIN) && (angle <= PHZ_0_BMAX) ) ) )
        {       /*对 BPSK/QPSK 统计+/-45°或者对 QPSK/L 统计 +/-180°的情况*/
                if(QPSKL_MODE==m_PSKmode)
                        temp = 3*PI2/4.0 - angle;
                else
                        temp = angle - 3*PI2/4.0;
                m_QFreqError = temp;
                if( m_PSKmode ) /*if QPSK */
                        temp = 280.0*fabs(temp);
                else
                        temp = 150.0*fabs(temp);
                if( temp < m_DevAve)
                        m_DevAve=   (1.0-1.0/SqTimeK)*m_DevAve + (1.0/SqTimeK)*temp;
                else
                        m_DevAve= (1.0-1.0/(SqTimeK*2.0))*m_DevAve
                                                + (1.0/(SqTimeK*2.0))*temp;
                if((m_OffCount > 20) ) {        /* fast squelch counter */
                        if( BPSK_MODE==m_PSKmode ) { /*对于 BPSK */
                                m_DevAve = 100.0 - 0.0;
                        }
                }
                else m_OffCount++;

                m_OnCount = 0;
                if( m_QFreqError >= 0.0 )、
                {
                        m_Pcnt++;
                        m_Ncnt = 0;
                }
                else
                {
                        m_Ncnt++;
                        m_Pcnt = 0;
                }
                if( (m_Pcnt<ELIMIT) && (m_Ncnt<ELIMIT) )
                        m_QFreqError = 0.0;
        }
}
if(m_OnCount >2)
        m_IMDValid = TRUE;
else
        m_IMDValid = FALSE;

if( m_AGCave > 10.0 )
{
```

```
            if( m_PSKmode ) /*对于 QPSK */
                m_SQLevel = 100 - (int)m_DevAve;
            else
                m_SQLevel = 100 - (int)m_DevAve;
            if( m_SQLevel >= m_SQThresh )
                m_SQOpen = TRUE;
            else
                m_SQOpen = FALSE;
        }
        else
        {
            m_SQLevel = 0;
            m_SQOpen = FALSE;
        }
        if(m_PSKmode)
        {
            if( m_QFreqError > .6 )/*  限制在  +/- 3 Hz */
                m_QFreqError = .6;
            if( m_QFreqError < -.6 )
                m_QFreqError = -.6;
        }
        else
        {
            if( m_QFreqError > 1.0 )/*限制在  +/- 5 Hz */
                m_QFreqError = 1.0;
            if( m_QFreqError < -1.0 )
                m_QFreqError = -1.0;
        }
        m_FperrAve = (1.0-1.0/m_NLPk)*m_FperrAve +
                        ( (1.0*PHZDERIVED_GN)/m_NLPk)*m_QFreqError;
    }
```

7.6 DSP 库简介

Microchip 公司将常用的数字信号处理函数做成库的形式，以方便用户使用。DSP 库主要提供了以下一些函数：

（1）矢量处理函数

- VectorMax：矢量最大值。
- VectorMin：矢量最小值。
- VectorCopy：矢量复制。
- VectorZeroPad：矢量填零
- VectorNegate：矢量取反。
- VectorScale：矢量加权计算。
- VectorAdd：矢量加。
- VectorSubtract：矢量减。

- VectorMultiply：矢量乘。
- VectorDotProduct：矢量点乘。
- VectorPower：矢量幂运算。
- VectorConvolve：矢量卷积运算。
- VectorCorrelate：矢量相关运算。

（2）加窗处理函数

- BartlettInit：Bartlett 加窗初始化。
- BlackmanInit：Blackman 加窗初始化。
- HammingInit: Hamming 加窗初始化。
- HanningInit：Hanning 加窗初始化。
- KaiserInit：Kaiser 加窗初始化。
- VectorWindow：矢量加窗。

（3）矩阵处理函数

- MatrixScale：矩阵加权运算。
- MatrixTranspose：矩阵转置。
- MatrixInvert：矩阵逆运算。
- MatrixAdd：矩阵加。
- MatrixSubtract：矩阵减。
- MatrixMultiply：矩阵乘。

（4）FIR 滤波器函数

- FIRDecimate：FIR 抽取滤波器。
- FIRInterpolate：FIR 插值滤波器。
- FIRLattice：FIR 格状滤波器。
- FIRLMS：FIR 的 LMS 算法滤波器。
- FIRLMSNorm：FIR 的归一化 LMS 算法滤波器。

（5）IIR 滤波器函数

- IIRTransposed：转置 IIR 滤波器。
- IIRLattice：格状 IIR 滤波器。

（6）常用变换函数

- BitReverseComplex：复数比特反转。
- FFTComplex：复数快速傅里叶变换。
- FFTComplexIP：原址（In-Place，IP）复数 FFT。
- IFFTComplex：复数快速傅里叶变换
- IFFTComplexIP：原址（In-Place，IP）复数 IFFT。
- DCT：离散余弦变换
- DCTIP：原址 DCT 算法。
- SquareMagnitudeCplx：复数的幅度平方。
- PIDCoeffCalc：PID 算法系数计算。
- PID：PID 算法。

附录　业余无线电简介

通常所说的业余无线电，是指业余业务和卫星业余业务，它们是国际电信联盟正式登记的 43 项无线电业务中的两种，业余无线电台管理是无线电管理的重要组成部分。

根据国际电信联盟的定义，业余业务是指供业余无线电爱好者进行自我训练、相互通信和技术研究的无线电通信业务。卫星业余业务是指利用地球卫星上的空间电台开展与业余业务相同目的的无线电通信业务。业余无线电爱好者是指经正式批准的。对无线电技术有兴趣的人，其兴趣纯系个人爱好而不涉及谋取利润。业余无线电台是指用于业余业务的电台，是开展业余业务和卫星业余业务所必需的一个或多个发射机、接收机，或者发射机与接收机的组合（包括附属设备）。

业余无线电台活动具有悠久的历史，可以追溯到 100 多年前无线电发明之日。无线电发明人马可尼就曾经说：“我本人就是一名业余爱好者。”世界发达国家都对业余无线电台活动采取了鼓励政策，国际电信联盟也为之分配了带宽多达 23GHz 以上的频率资源。

业余业务是拥有台站数量最多的无线电业务。全世界的业余无线电台总数达到了 300 万个，其中美国和日本约占 1/3 强。值得一提的是，在全世界业余无线电台发展趋于平缓的情况下，我国业余无线电台数量持续加速增加。目前持有无线电台执照的业余无线电爱好者的数量估计为 4 万。世界各地的无线电爱好者以技术研究为主题，以“体谅、忠诚、进取、友爱、适度、爱国”为原则，通过不同地区电台之间的联系研究电波传播规律，增进相互间的友谊，为国家和社会服务。

1．业余无线电的主要活动形式

业余无线电台爱好者的日常活动包括自我训练、技术研究、互相通信等。自我训练包括基本联络技巧的训练和参加各类竞赛活动；技术研究包括装备改进、通信模式研究等；互相通信包括与其他爱好者之间的联系、救灾应急通信以及其他公益服务。

国际上比较常见的活动内容大致有以下几种：

远距离通信（DX）、竞赛（Contest）、远征（Dxpedition）、信标（Beacon Project）、V/UHF 特殊传播通信等。也可分为模拟通信（如 SSB、FM 话等）和数字通信（如 CW、RTTY、PACKET 等）。

空间通信也是业余无线电活动的一项主要内容，业余卫星通信、地－月－地（EME）通信、散射通信（如电离层散射通信、流星余迹散射通信等）、空间飞行器通信等均是业余无线电爱好者涉猎的范畴。我国也发射了一颗名为“希望一号”的业余卫星，为广大业余无线电爱好者提供了开放的试验平台，如图 1 所示。

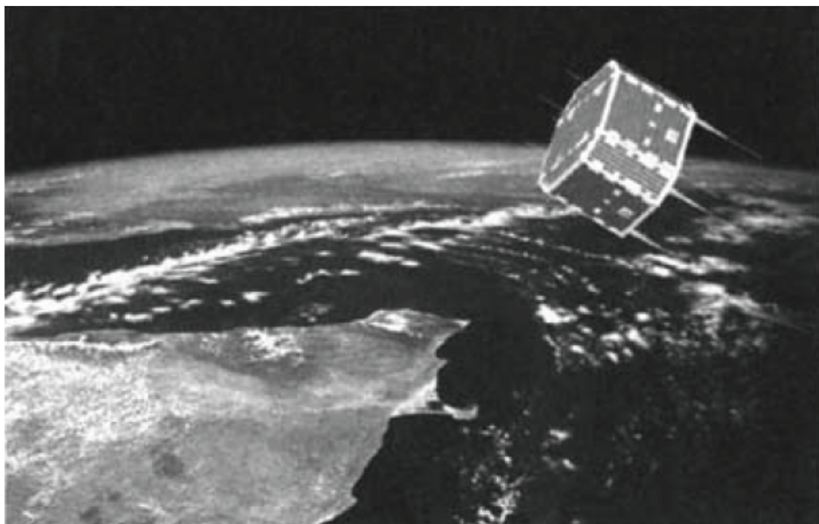


图 1 “希望一号”卫星

除上所述外，宇宙观测（射电天文）、无线电技术工程（Radio Engineer）、自制器材设备、天线设计、软件及接口设计、新频段和新调制方式研究（如微波通信、扩展频谱通信等）、无线电测向及定位、快速收发报和社会服务等，也是业余无线电活动所涉及的内容。

业余无线电台活动除进行日常机上通信、交换联络卡片外，还经常举办各类竞赛。业余无线电台的竞赛，一般是自发性的，参加者多以娱乐为主要目的。一般的比赛规则是在规定的时间内，看谁联络的电台多、距离远、范围广。其记分的方式大多数是以所联络的电台作为基本分（距离近的得分少，距离远的得分多），以这些电台的分布范围（如不同国家、不同所在分区、不同的呼号前缀等）为系数分，两者相乘，以总分的多少计算成绩。

竞赛的呼叫，一般是在 CQ 后加“CONTEST”（话）或是“TEST”（报）。在竞赛中，联络双方必须按比赛规则的规定交换特定的竞赛信息，内容一般由信号报告再分别加上本台的分区数或者竞赛中已联络次数的序号、发射功率等组成。为了争取速度和不影响别人，竞赛中的联络要尽量简洁，除了交换的报告外，一般不谈其他的内容。竞赛的时间一般都设在周末．持续时间多为 24 小时或 48 小时。每个比赛都设有单波段/多波段、单人/多人、单发信机/多人多发信机，及 QRP 小功率等不同项目的奖励。表 1 列出了目前较为流行的业余无线电活动。

表 1　业余无线电的主要活动

名　　称	主　办　者	时　　间
CQWW 远距离世界比赛	美国 CQ 杂志	每年 9 月最后一个周末为电传 RTTY 项目比赛，10 月最后一个周末为话项目的比赛，11 月最后一个周末为报项目的比赛
CQ WPX 世界比赛	美国 CO 杂志	每年 3 月的最后一个周末进行单边带话项目的比赛，5 月最后一个周末为报项目的比赛
IARU 短波世界锦标赛	国际业余无线电联盟（IARU）	每年 7 月份的第二个周末举行
全亚洲比赛	日本业余无线电联盟	每年 6 月的第三个周末进行 CW 项目的比赛，每年 9 月的第一个周末进行话项目比赛
自制小功率发射机夏季 CW 比赛	美国 ARCI 俱乐部	每年 7 月第三周周日 20:00 UTC 到周二 00:00 UTC 举行

2．业余业务使用的频率和功率

设置和操作业余无线电台的业余无线电爱好者必须经过相应的培训，考取操作证书。业余无线电台使用的频率，均需遵守无线电主管部制定的频率划分规定。表 2 给出了目前常用的业余无线电频段。

表 2　业余无线电频段

频　段	业务地位	频　段	业务地位
1.800MHz～2.000MHz	主要业余（共用）	2300MHz～2450MHz	次要业务
3.500MHz～3.900MHz	主要业余（共用）	3300MHz～3500MHz	次要业务
7.000MHz～7.100MHz	专用	5650MHz～5850MHz	次要业务
7.100MHz～7.200MHz	专用（从 2009 年 3 月 29 日以后）	10.0GHz～10.5GHz	次要业务
10.100MHz～10.150MHz	次要业务	24.0GHz～24.05GHz	主要业余（共用）
14.000MHz～14.250MHz	专用	24.05GHz～24.25GHz	次要业务
14.250MHz～14.350MHz	主要业余（共用）	47.0GHz～47.2GHz	专用
18.068MHz～18.168MHz	主要业余（共用）	76.0GHz～77.5GHz	次要业务
21.000MHz～21.450MHz	专用	77.5GHz～78.0GHz	主要业务
24.890MHz～24.990MHz	主要业余（共用）	78.0GHz～81.0GHz	次要业务
28.000MHz～29.700MHz	专用	122.25GHz～123GHz	次要业务
50.000MHz～54.000MHz	主要业余（共用）	134.0GHz～136.0GHz	主要业务
144.000MHz～146.000MHz	主要业务	136.0GHz～141.0GHz	次要业务
146.000MHz～148.000MHz	主要业余（共用）	241.0GHz～248.0GHz	次要业务
430.000MHz～440.000MHz	次要业务	248.0GHz～250.0GHz	主要业务
1240MHz～1300MHz	次要业务		

截至 2007 年世界无线电大会前，中国无线电主管部门为业余业务和卫星业余业务分配了 23 个频段，这些频段分布在 1.8MHz～250GHz 之间，部分频段中业余业务作为主要业务使用。在 2007 年世界无线电大会上，各国无线电主管部门通过讨论，最终通过了决议，在 135.7kHz～137.8kHz 频段为业余业务开辟新的频段，并在三区将 7.100～7.200MHz 频段划归业余业务使用。

我国无线电主管部门对不同等级的业余无线电爱好者可用的最大功率分别作了规定（见表 3）。随着新技术的不断发展，业余无线电爱好者对功率的需求也在不断提高，这也要求无线电主管部门与时俱进，适时考虑出台新的政策。

表 3　最大可用功率的规定

操作等级	300MHz 以下/W	30MHz 以上/W
一级	500	25
二级	100	25
三级	25	25
四级	10	5
五级	——	——

3．业余无线电台的通信距离有多远

业余无线电台的通信距离有多远呢？可能很多人最先想到的就是这个问题。一般情况

下，对于短波频段的通信，直射波及地波传播时一般可以在 300km 以内通信（150～1000km），天波传播时可以实现全球通信（150～1000km），不过这在很大限度上取决于电离层的情况。在超短波频段（VHF/UHF）通信时，直射波传播可以实现视距通信（5～30W）。如果通过中继则可以实现城市间、省内的通信。如果采用新的数字通信和互联网平台，则可以实现省际甚至是国际间的通信。

值得一提的是，与其他的通信方式不同，业余无线电台间的通信一般不追求稳定的联通概率，甚至于有很大一部分业余无线电爱好者还在追求某种特殊传播条件下的异常传播，某一次特殊情况下的超远距离传播对业余无线电台来说可能具有更大的意义。在某些特殊传播条件下（如突发电离层、大气管道、赤道共轭等），不通过中继台，超短波电台也可以实现 2000km 或更远的远距离通信（300～1000km）。如果通过流星余迹散射、对流层散射、月面反射、业余卫星等通信技术，超短波电台更可以实现洲际通信。

由此可见，与其他业务不同，业余业务天生就具有跨省和涉外联络的特性。业余无线电台活动曾经为很多国家培养了大量肯钻研、能动手的无线电和电子技术人才，成为这些国家经济发展的宝贵人力资源。广泛普及的业余无线电台活动还形成了雄厚的民间通信资源储备，在突发重大灾害时为社会提供了有效的应急通信服务。作为无线电管理部门，应重视业余业务和卫星业余业务，以促进其合理发展。

4．常见业余无线数字通信方式

最近几十年，业余无线电数字通信有了飞速的发展。从二战结束到 20 世纪 80 年代初期，业余无线电爱好者能够使用的唯一一种短波数字通信模式是 RTTY（无线电传打字）。1983 年，AMTOR 首次出现。巧合的是，深受大众欢迎的个人计算机也是这一年出现的。AMTOR 是第一种具有纠错功能的数字通信模式。

从 20 世纪 80 年代开始，数字通信技术的发展明显加快。20 世纪 80 年代中期，分组通信出现，在很长的一段时间内，成为数字通信的主流。随着微处理器技术的发展，又有一些新的数字通信模式出现，包括 Clover、PACTOR 和 G-TOR，这些模式能在弱信号、强干扰等恶劣情况下工作，并且具有纠错功能。20 世纪 90 年代末期，一种高度依赖计算机的新数字通信模式出现了，它就是 PSK31。下面就对目前比较流行的三种业余无线电数字通信方式进行简单介绍。

（1）RTTY

RTTY（无线电传打字）是最古老的 HF 数字通信模式，已经有 50 多年的历史。但是现在仍然有许多业余无线电在使用，主要有两个原因。

1）RTTY 设备比较简单。进行 RTTY 通信，只需要两个设备：一个多模式控制器（或者一个有声卡的计算机），一个 SSB 电台。

2）RTTY 通信容易操作。RTTY 通信不需要握手信号，因此电台之间不需要复杂的协议，只要会键盘打字即可操作。

现在 HF 的 RTTY 通信主要出现在 20m 波段上，在 40m 和 15m 波段上偶尔也可以听到 RTTY 通信。

利用 RTTY 模式，可以传送、接收各种字符。每个 RTTY 字符由 5 个二进制位组成。二进制 1 叫做 mark，通常用 2125Hz 的频率代表，二进制 0 叫做 space，通常用 2295Hz 的频率代表。在每个字符的前面有一个开始脉冲。RTTY 通信的标准传送速率是每分钟 60 个词

（每个词 5 位），相当于 45 波特。

在 RTTY 通信中，采用 5 位编码的博多码（Baudot）来表示 26 个大写字母、10 个数字以及一组标点符号。为了解决 5 位编码太少的问题，在字符中插入了 LTRS（11111）和 FIGS（11011）两个特殊符号来表示当前的状态。LTRS 表示当前处在字符状态，它后面的所有字符都是字母；FIGS 表示当前处在图形状态，它后面的所有字符都是数字和标点符号。

每个 RTTY 解码器都包括 mark/space 信号滤波器。解码器越灵敏，选择性越高，RTTY 的通信质量越好。尤其是在干扰严重、信号微弱的情况下，解码器的灵敏度与选择性显得尤为重要。

（2）PSK31

PSK31（31.25Baud 的 PSK 信号）是彼得・马丁内斯（G3PLX）的发明，他同时也是 AMTOR 的发明者。开始的时候彼得希望发明一种新型的数字通信模式，既能像 RTTY 那样容易使用，又能在微弱信号情况下正常使用。此外，彼得还考虑到带宽的问题。HF 数字通信的带宽通常比较窄，在某些时候（例如竞赛时段），显得比较拥挤。彼得决定在现在的基础上，发明一种具有弱信号通信能力的数字模式。

PSK31 的试验开始于 20 世纪 90 年代中期，当时的业余无线电爱好者还在使用 DOS 操作系统和 DSP 开发平台。1999 年彼得开发出了第一个基于 Windows 操作系统的 PSK31 软件，它需要一个配有声卡的计算机。随着带有声卡的计算机的普及，PSK31 通信模式开始流行起来了。

PSK31 中的 PSK 是英文 Phase Shift Keying 的缩写，中文意思是相移键控；31 表示这种模式的传输速率是 31bit/s。严格来说 PSK31 的传输速率是 31.25 bit/s。

在 RTTY 中，采用的是博多码，每个字母、数字或标点符号采用固定的 5 个二进制位表示。在 PSK31 中彼得采用了一种新编码——Varicode（可变长编码）。“可变”的意思是：每个字符所使用的二进制位数不是固定的，而是可变的，彼得参考莫尔斯电码，将较短的编码分配给英文中常用的字母，将较长的编码分配给不常用的字母。如小写 e 是英文中最常用的字母，所以分配给它的编码是 11，小写 z 是最不常用的字母，所以它的编码是 111010101。

与莫尔斯电码和 RTTY 一样，PSK31 字符之间也需要一个间隔。可变长编码使用 00 作为间隔信号。彼得在为每个字符指定可变长编码的时候，尽量避免出现两个连续的 0，因此在每个字符的可变长编码中都没有 00。

在 PSK31 中，采用 BPSK（二相相移键控）信号来进行调制。彼得把 0 相分配给 1，180°相移分配给 0。发送方在发送 PSK31 信号之前，会发送一组二进制 0 作为同步信号，接收端收到同步信号之后，马上进入同步状态，然后就可以进行正确解调了。

PSK31 将窄带技术、DSP 技术、同步解调技术结合起来，即使在微弱信号下，也能正确接收信号。PSK31 的弱信号处理能力完全能和 CW 媲美，性能明显优于 RTTY。

目前常用的 PSK31 信号集中在 14070kHz 附近，此外在下列频点附近也可以发现 PSK31 通信：3580kHz、7070kHz、10140kHz、21070kHz 和 28120kHz。

PSK31 信号的声音很独特，不同于在业余段听到的其他数字信号。如果你听到的是叽叽喳喳的声音，那很可能是 G-TOR 信号，而不是 PSK31 信号。PSK31 信号是一种类似于鸟鸣的声音，需要多听几次才能熟悉。

PSK31 是 PSK 模式中使用最广泛的一种，除此之外还有其他模式。如 PSK63/125，它

们是用于高速数据交换的，当然它们的带宽也相应增加了。

（3）PACTOR

1991 年，汉斯 • 彼得 • 海尔费特（DL6MAA）和乌尔里希 • 斯特拉特（DF4KV）开发出 PACTOR。十几年来，PACTOR 一直是最流行的脉冲式 HF 数字通信模式。PACTOR 有 3 个版本，分别是 PACTOR I、PACTOR II 和 PACTOR III。目前最常用的是 PACTOR II/III。PACTOR 之所以被称为脉冲式模式，是因为它发送数据的方式特殊。RTTY、PSK31、MFSK16 等模式在工作的时候，发送连续的数据流，而 PACTOR 在工作的时候，以脉冲的形式发送不连续的小数据包。接收方收到一个数据包后，如果数据包没有错误，给发送方发送 ACK（确认）信号，请求发送下一个数据包；如果数据包有错误，给发送方发送 NAK（否定确认）信号，请求重新发送这个数据包。这种一来一往的双向对话发出类似蟋蟀的鸣叫声，其中时间较长的声音是传送数据包的声音，时间较短的声音是传送 ACK 和 NAK 信号的声音。

传统的脉冲方式（例如 AMTOR），为了纠错，会反复发送同一个数据包，直到这个数据包没有错误为止。这种做法降低了通信速率，尤其在干扰严重的情况下，通信速率极低。

PACTOR 采用类似的纠错方式。每个数据包发送之后，发送方都会收到接收方的确认信号，如果收到 ACK 信号，表示数据包没有错，继续发送下一个数据包；如果收到了 NAK 信号，表示数据包有错误，需要将当前数据包重新发一遍。从表面上看，PACTOR 和 AMTOR 的纠错机制好像没有区别，但实际上 PACTOR 有一个很大的特点，就是 PACTOR 采用了先进的记忆式 ARQ。在记忆式 ARQ 中，当接收端收到一个错误的数据包后，首先分析这个数据包的各个片段是否有错（假设数据包有 10 个片段），然后将正确的片段存储起来（假设存储了 1/3/5/7/9 片段）。再次接收的时候，可能就会得到正确的其他片段，这样重复多次后就能得到正确的数据包了。

PACTOR 采用哈夫曼编码方式，字符的二进制位数明显缩短，提高了通信效率。

要进行 PACTOR 通信并不复杂，通常需要下列硬件与软件：

- 一部 SSB 电台。PACTOR 使用类似于 RTTY 的 mark/space 系统。可以通过电台的话筒插座（或附属插座），以 AFSK 模式发送音频信号，也可以用 FSK 模式，让发射机生成音频信号。
- 一个 PACTOR 终端软件。
- 一台支持 PACTOR 通信的多模式通信处理器（MCP）。

注意，不能用声卡来代替 MCP，只能采用 MCP 来进行 PACTOR 通信。

参 考 文 献

[1] Steve Ford．HF/VHF 数字通信手册[M]．张宏，译．北京：人民邮电出版社，2010．

[2] 沈越泓，高媛媛，魏以民．通信原理[M]．2 版．北京：机械工业出版社，2008．

[3] 王金龙，等．无线通信系统的 DSP 实现[M]．北京：人民邮电出版社，2002．

[4] 西瑞克斯（北京）通信设备有限公司．无线通信的 MATLAB 和 FPGA 实现[M]．北京：人民邮电出版社，2009．

[5] 尹虎. 什么是业余无线电[J]．中国无线电，2009（12）．

ISBN 978-7-111-36505-1

封面设计：子时文化 ZiShi Culture



Wireless Communication System Design Based on dsPIC

基于dsPIC的无线通信系统设计

内容简介

本书采用美国微芯公司的dsPIC芯片来实现无线通信中的常见算法，并通过剖析一个典型案例来分析在业余无线电爱好者中广泛应用的NUE-PSK调制解调器，从而展示出dsPIC在无线通信中的典型应用。本书分7章，内容包括数字信号控制器及其在无线通信中的应用、MPLAB C30编译器、数字滤波器的设计与实现、数字调制解调器的设计与实现、同步功能的设计与实现、信道编/译码器的设计与实现、基于dsPIC的无线通信设备NUE-PSK实例剖析。

本书可以作为通信与信息系统专业研究生、本科生的参考书，也可供通信工程技术人员参考。

地址：北京市百万庄大街22号
电话服务
社服中心：(010)88361066
销售一部：(010)68326294
销售二部：(010)88379649
读者购书热线：(010)88379203

邮政编码：100037
网络服务
门户网：http:www.cmpbook.com
教材网：http:www.cmpedu.com
封面无防伪标均为盗版

定价：32.00 元

上架建议 通信技术/无线通信

ISBN 978-7-111-36505-1

